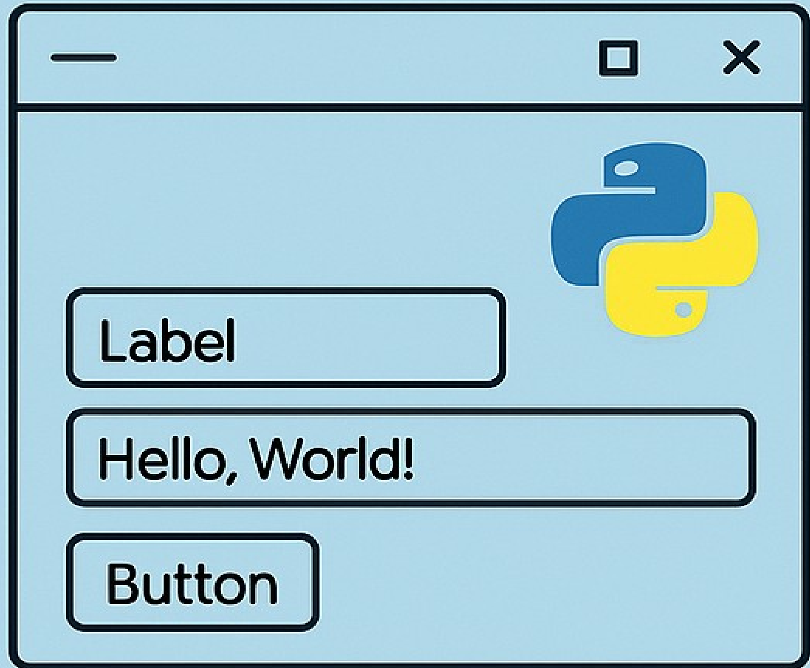
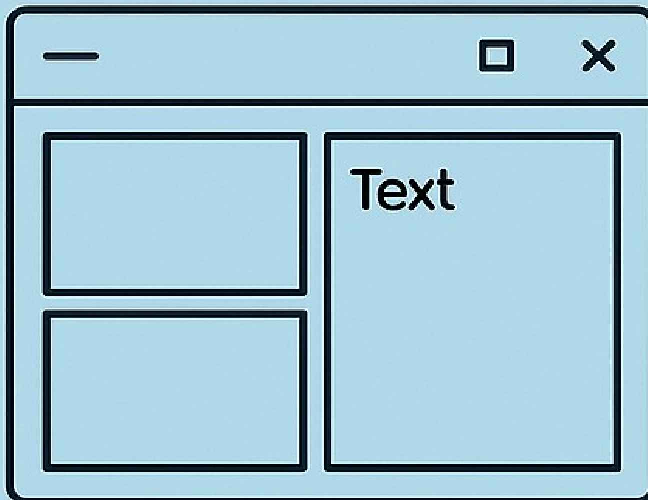


TKINTER



LUIS MINGUILLÓN PASCUAL

TKINTER

LUIS MINGUILLON PASCUAL

Luis Minguillón Pascual

Prefacio

He escrito este libro movido por una convicción profunda: el saber nos hará libres. No creo que el conocimiento deba ser un privilegio reservado a unos pocos, sino una herramienta que se comparte, que se multiplica y que tiene sentido solo cuando se pone al servicio de los demás. Por eso, cada línea de este libro nace de la idea de que el conocimiento no se guarda: se entrega. Se comparte para crecer, para avanzar y, sobre todo, para ser más libres.

Tkinter, el módulo gráfico estándar de Python, ha sido durante años una herramienta valiosa pero injustamente ignorada. A pesar de su potencia y sencillez, la información disponible sobre él es escasa, dispersa y a menudo poco clara. Esa carencia fue una de las razones que me llevó a emprender esta tarea: hacer un libro claro, completo y accesible para quienes, como yo, creen que aprender es también una forma de emanciparse.

Este libro no nace desde la autoridad académica ni desde el pedestal del experto, sino desde la voluntad de ayudar, de aprender mientras se enseña, y de acompañar a otros en el camino del conocimiento. Aquí encontrarás explicaciones detalladas, ejemplos útiles y una guía honesta construida desde la experiencia y la pasión.

No quiero terminar este prefacio sin dejar claro algo importante: **no agradezco el apoyo recibido**. No porque no lo haya habido, sino porque este camino lo he recorrido en soledad, con esfuerzo y decisión. Sin embargo, no puedo dejar de mencionar con cariño a una persona muy especial: **mi mujer, Marisa**. A ella, por estar, por ser, y por acompañarme, va dedicado este trabajo.

INDICE TKINTER

- 6.3 Eventos predefinidos
- 6.4 Eventos personalizados
- 6.5 Eventos múltiples en un solo widget
- **Lección 7: Variables de Control en Tkinter**
    - 7.1 StringVar, IntVar, DoubleVar, BooleanVar
    - 7.2 Enlazar variables a widgets
    - 7.3 Actualización automática
- **Lección 8: Menús y Barras en Tkinter**
    - 8.1 Menú principal (Menu)
    - 8.2 Submenús
    - 8.3 Separadores
    - 8.4 Atajos de teclado en menús
    - 8.5 Menús emergentes (popup)
- **Lección 9: Cuadros de Diálogo en Tkinter**
    - 9.1 messagebox: información, advertencia, error
    - 9.2 filedialog: abrir y guardar archivos
    - 9.3 colorchooser: seleccionar color
    - 9.4 simpledialog: entradas simples del usuario
- **Lección 10: Canvas y Gráficos en Tkinter**
    - 10.1 Crear un lienzo (Canvas)
    - 10.2 Dibujo de líneas, óvalos, rectángulos, polígonos, texto
    - 10.3 Movimiento de objetos
    - 10.4 Detección de colisiones y coordenadas
    - 10.5 Imágenes en el lienzo
- **Lección 11: Trabajar con Imágenes en Tkinter**
    - 11.1 Cargar imágenes con PhotoImage
    - 11.2 Formatos compatibles
    - 11.3 Mostrar imágenes en widgets
    - 11.4 Uso de PIL (Pillow) para formatos avanzados
- **Lección 12: Widgets Avanzados (ttk)**
    - 12.1 Introducción a ttk (Themed Tk)
    - 12.2 ttk.Label, ttk.Entry, ttk.Button, ttk.Checkbutton
    - 12.3 Combobox
    - 12.4 Notebook (pestañas)
    - 12.5 Treeview (listas y jerarquías)
    - 12.6 Progressbar
    - 12.7 Separator
    - 12.8 Style: personalización de temas y estilos
- **Lección 13: Validación de Entradas en Tkinter**
    - 13.1 Métodos de validación (validate, validatecommand)
    - 13.2 Restricción de datos (números, letras, longitudes)
    - 13.3 Validación con expresiones regulares
- **Lección 14: Temporizadores y Animaciones en Tkinter**

Además de las lecciones, el documento incluye las siguientes secciones prácticas:

- **30 Ejercicios Básicos de Tkinter en Python con su Solución y Explicación**
- **Programa completo en Tkinter: Agenda completa json en tkinter**

Lección: Introducción a Tkinter

1.1 ¿Qué es Tkinter?

Tkinter es el módulo estándar de Python para la creación de interfaces gráficas de usuario (GUI). Permite desarrollar aplicaciones con ventanas, botones, cuadros de texto, menús, entre otros elementos interactivos. Tkinter es un contenedor que proporciona acceso a la biblioteca Tcl/Tk, permitiendo construir GUIs de manera sencilla usando Python.

1.2 Historia y origen

Tkinter se basa en la biblioteca Tcl/Tk, desarrollada a finales de los años 80 por John Ousterhout. Tcl es un lenguaje de scripting, y Tk es su toolkit gráfico. Python integró Tkinter como su interfaz oficial para GUIs desde la versión 1.5.2. A lo largo de los años, Tkinter ha evolucionado para ofrecer más widgets, mayor compatibilidad y mejoras visuales. Aunque existen otras bibliotecas como PyQt o wxPython, Tkinter sigue siendo muy utilizado por su simplicidad y porque viene incluido con Python.

1.3 Ventajas y limitaciones

Ventajas:

- Está incluido en la instalación estándar de Python, no requiere instalación adicional.
- Es fácil de aprender y usar, especialmente para principiantes.
- Tiene suficiente funcionalidad para desarrollar aplicaciones de escritorio básicas o intermedias.
- Cuenta con buena documentación y una comunidad activa.

Limitaciones:

- Estéticamente puede parecer anticuado frente a bibliotecas más modernas.
- No está pensado para aplicaciones muy complejas ni con interfaces gráficas avanzadas.
- Es menos flexible que frameworks como PyQt o Kivy en lo visual.
- El diseño de interfaces no es responsivo por defecto.

1.4 Instalación y configuración del entorno

Tkinter generalmente viene preinstalado con Python. Para verificar si está disponible, puedes ejecutar el siguiente código en una consola de Python:

```
import tkinter
print("Tkinter está disponible")
```

Si obtienes un error, significa que necesitas instalarlo. En la mayoría de sistemas Linux puedes instalarlo con:

En Ubuntu/Debian:
sudo apt-get install python3-tk

En Fedora:
sudo dnf install python3-tkinter

En Windows y macOS no suele requerirse instalación adicional si se ha instalado Python desde la web oficial.

Para desarrollar con Tkinter, se puede usar cualquier entorno de desarrollo como IDLE, VSCode, PyCharm o simplemente un editor de texto.

1.5 Estructura básica de un programa con Tkinter

Un programa simple en Tkinter sigue esta estructura básica:

```
import tkinter as tk

# Crear la ventana principal
ventana = tk.Tk()
ventana.title("Mi primera GUI")
ventana.geometry("300x200")

# Crear un widget (por ejemplo, una etiqueta)
etiqueta = tk.Label(ventana, text="¡Hola, mundo!")
etiqueta.pack()

# Iniciar el bucle principal
ventana.mainloop()
```

Explicación:

- Se importa el módulo tkinter con un alias (tk).
- Se crea la ventana principal mediante `tk.Tk()`.
- Se definen los widgets y se añaden a la ventana usando métodos como `pack()`.
- Finalmente, se llama a `mainloop()` para mostrar la ventana y esperar interacciones del usuario.

1.6 Compatibilidad entre versiones de Python (2.x vs 3.x)

En Python 2, Tkinter se importaba con una "T" mayúscula:

```
import Tkinter as tk
```

En Python 3, se cambió a minúsculas:

```
import tkinter as tk
```

Este es uno de los principales cambios de compatibilidad. Además, en Python 3 se han mejorado algunas funciones y se han corregido errores. En general, se recomienda usar siempre Python 3 para aprovechar mejoras de seguridad, compatibilidad y soporte a largo plazo.

Tkinter en Python 3 es más estable y es la versión más utilizada actualmente para el desarrollo de interfaces gráficas. Algunas bibliotecas complementarias como ttk (Themed Tkinter) también se integran mejor en Python 3.

Lección 2: Ventana Principal (Tk)

2.1 Crear una ventana principal
La ventana principal en una aplicación Tkinter se crea instanciando la clase Tk. Esta clase representa la raíz de la interfaz gráfica, y solo puede haber una instancia de Tk por programa.
Ejemplo:

```
import tkinter as tk

ventana = tk.Tk()
```

2.2 Configurar tamaño, título y posición
Una vez creada la ventana, se puede configurar su tamaño, título y posición inicial.

- Título: se configura con el método title
- Tamaño: se puede establecer con geometry
- Posición: también se establece con geometry, en formato "anchoxalto+x+y"

Ejemplo:

```
ventana.title("Mi Aplicación")              # Cambiar título
ventana.geometry("400x300+100+100")         # 400x300 píxeles, posición (100,
100)
```

2.3 Métodos comunes (geometry, resizable, iconbitmap, etc.)
Tkinter ofrece varios métodos útiles para modificar el comportamiento y apariencia de la ventana principal:

- geometry(cadena): establece el tamaño y posición
- resizable(ancho, alto): permite o impide cambiar el tamaño en horizontal o vertical
- iconbitmap(ruta): cambia el ícono de la ventana (solo .ico en Windows)
- config(**opciones): permite cambiar opciones como el fondo con `bg`
- maxsize(ancho, alto): limita el tamaño máximo
- minsize(ancho, alto): limita el tamaño mínimo

Ejemplo:

```
ventana.resizable(False, False)              # Ventana no redimensionable
ventana.iconbitmap("icono.ico")              # Cambia el ícono
ventana.config(bg="lightblue")               # Cambia el fondo
ventana.maxsize(600, 400)                    # Tamaño máximo permitido
ventana.minsize(200, 150)                    # Tamaño mínimo permitido
```

2.4 Ciclo de eventos (mainloop)
Toda aplicación Tkinter necesita un ciclo de eventos, que se activa mediante el método mainloop. Este ciclo mantiene la ventana abierta y responde a eventos como clics o teclado. Debe colocarse al final del programa.
Ejemplo:

```
ventana.mainloop()
```

2.5 Salir del programa (destroy, quit)

Hay dos formas comunes de cerrar la ventana principal:

- destroy(): cierra la ventana y libera los recursos asociados
- quit(): finaliza el ciclo de eventos pero no destruye completamente la ventana

Ejemplo:

```
ventana.destroy()  # Se puede usar en un botón o evento
ventana.quit()     # Finaliza el mainloop, pero no cierra la ventana
inmediatamente
```

Para salir desde un botón:

```
boton_salir = tk.Button(ventana, text="Salir", command=ventana.destroy)
boton_salir.pack()
```

Lección 3. Widgets Básicos en Tkinter

3.1 Label
El widget Label se utiliza para mostrar texto o imágenes que no se pueden editar. Se usa comúnmente para mostrar información al usuario.
Ejemplo:

```
from tkinter import *
root = Tk()
etiqueta = Label(root, text="Hola, mundo")
etiqueta.pack()
root.mainloop()
```

3.2 Button
El widget Button se utiliza para ejecutar una acción cuando el usuario hace clic sobre él. Puede estar asociado a una función.
Ejemplo:

```
from tkinter import *
def saludar():
    print("¡Hola!")
root = Tk()
boton = Button(root, text="Saludar", command=saludar)
boton.pack()
root.mainloop()
```

3.3 Entry
El widget Entry permite al usuario introducir una sola línea de texto.
Ejemplo:

```
from tkinter import *
root = Tk()
entrada = Entry(root)
entrada.pack()
root.mainloop()
```

3.4 Text
El widget Text permite ingresar o mostrar texto de varias líneas.
Ejemplo:

```
from tkinter import *
root = Tk()
texto = Text(root, height=5, width=30)
texto.pack()
root.mainloop()
```

3.5 Checkbutton
El widget Checkbutton permite seleccionar o deseleccionar una opción. Se puede usar para opciones booleanas.
Ejemplo:

```
from tkinter import *
root = Tk()
var = IntVar()
check = Checkbutton(root, text="Aceptar términos", variable=var)
check.pack()
root.mainloop()
```

3.6 Radiobutton

El widget Radiobutton permite seleccionar solo una opción de un conjunto.

Ejemplo:

```
from tkinter import *
root = Tk()
var = IntVar()
opcion1 = Radiobutton(root, text="Opción 1", variable=var, value=1)
opcion2 = Radiobutton(root, text="Opción 2", variable=var, value=2)
opcion1.pack()
opcion2.pack()
root.mainloop()
```

3.7 Listbox

El widget Listbox se utiliza para mostrar una lista de elementos. El usuario puede seleccionar uno o varios.

Ejemplo:

```
from tkinter import *
root = Tk()
lista = Listbox(root)
lista.insert(1, "Python")
lista.insert(2, "Java")
lista.insert(3, "C#")
lista.pack()
root.mainloop()
```

3.8 Spinbox

El widget Spinbox permite seleccionar un valor numérico o de una lista usando flechas.

Ejemplo:

```
from tkinter import *
root = Tk()
spin = Spinbox(root, from_=0, to=10)
spin.pack()
root.mainloop()
```

3.9 Scale

El widget Scale permite seleccionar un valor numérico arrastrando una barra deslizante.

Ejemplo:

```
from tkinter import *
root = Tk()
escala = Scale(root, from_=0, to=100, orient=HORIZONTAL)
escala.pack()
root.mainloop()
```

3.10 Message

El widget Message es similar al Label, pero adapta el texto a varias líneas automáticamente.

Ejemplo:

```
from tkinter import *
root = Tk()
mensaje = Message(root, text="Este es un mensaje de varias líneas que se ajusta
al tamaño del widget.")
mensaje.pack()
root.mainloop()
```

3.11 Menú desplegable (OptionMenu)

El widget OptionMenu permite seleccionar una opción de una lista desplegable.

Ejemplo:

```
from tkinter import *
root = Tk()
opcion = StringVar()
opcion.set("Python")
menu = OptionMenu(root, opcion, "Python", "Java", "C++")
menu.pack()
root.mainloop()
```

Lección 4: Widgets Contenedores en Tkinter

En esta lección estudiaremos los widgets contenedores que permiten organizar y estructurar visualmente otros widgets dentro de una aplicación Tkinter. Los principales contenedores que ofrece Tkinter son: Frame, LabelFrame, PanedWindow, Toplevel y Canvas.

4.1 Frame

El widget Frame es un contenedor básico que se utiliza para agrupar otros widgets. No tiene bordes visibles por defecto, pero se puede personalizar para mostrar bordes o fondo de color.

Sintaxis:
frame = tk.Frame(master, opciones)

Opciones comunes:

- bg: color de fondo
- width, height: dimensiones
- relief: tipo de borde (flat, raised, sunken, groove, ridge)
- borderwidth: grosor del borde

Ejemplo:
```
import tkinter as tk

root = tk.Tk()
frame = tk.Frame(root, bg="lightblue", width=200, height=100, relief="ridge", borderwidth=5)
frame.pack(padx=10, pady=10)

label = tk.Label(frame, text="Este es un Frame")
label.pack()

root.mainloop()
```

4.2 LabelFrame

LabelFrame es un tipo especial de Frame que permite añadir un título visible. Sirve para agrupar widgets relacionados bajo un encabezado descriptivo.

Sintaxis:
labelframe = tk.LabelFrame(master, text="Texto", opciones)

Opciones adicionales:

- text: el texto del encabezado
- labelanchor: posición del texto (n, ne, e, se, s, sw, w, nw, center)

Ejemplo:
```
import tkinter as tk

root = tk.Tk()
lf = tk.LabelFrame(root, text="Datos personales", padx=10, pady=10)
lf.pack(padx=10, pady=10)
```

```
tk.Label(lf, text="Nombre:").pack()
tk.Entry(lf).pack()

tk.Label(lf, text="Edad:").pack()
tk.Entry(lf).pack()

root.mainloop()
```

4.3 PanedWindow

PanedWindow es un contenedor que permite añadir widgets en una disposición horizontal o vertical con divisores ajustables entre ellos. Estos divisores permiten al usuario redimensionar las áreas contenidas.

Sintaxis:
paned = tk.PanedWindow(master, orient=tk.HORIZONTAL|tk.VERTICAL, opciones)

Método importante:

- add(widget): añade un widget al panel

Ejemplo:

```
import tkinter as tk

root = tk.Tk()
pw = tk.PanedWindow(root, orient=tk.HORIZONTAL)
pw.pack(fill=tk.BOTH, expand=True)

left = tk.Label(pw, text="Panel Izquierdo", bg="lightgray")
pw.add(left)

right = tk.Label(pw, text="Panel Derecho", bg="lightblue")
pw.add(right)

root.mainloop()
```

4.4 Toplevel

Toplevel se usa para crear ventanas secundarias independientes del `root`, útiles para cuadros de diálogo, ventanas de configuración, etc.

Sintaxis:
top = tk.Toplevel(master, opciones)

Opciones comunes:

- title: título de la ventana
- geometry: dimensiones iniciales

Ejemplo:

```
import tkinter as tk

def abrir_ventana():
nueva = tk.Toplevel()
nueva.title("Ventana secundaria")
```

```
nueva.geometry("200x100")
tk.Label(nueva, text="Esta es una nueva ventana").pack(padx=10, pady=10)

root = tk.Tk()
tk.Button(root, text="Abrir ventana", command=abrir_ventana).pack(padx=20, pady=20)
root.mainloop()
```

4.5 Canvas

Canvas es un widget poderoso para crear gráficos, dibujos, formas personalizadas, imágenes y animaciones.

Sintaxis:
canvas = tk.Canvas(master, opciones)

Opciones comunes:

- width, height: tamaño del canvas
- bg: color de fondo

Métodos útiles:

- create_line(x1, y1, x2, y2, opciones): dibuja una línea
- create_rectangle(x1, y1, x2, y2, opciones): dibuja un rectángulo
- create_oval(x1, y1, x2, y2, opciones): dibuja una elipse
- create_text(x, y, text="texto", opciones): añade texto
- create_image(x, y, image=imagen): muestra una imagen

Ejemplo:

```
import tkinter as tk

root = tk.Tk()
canvas = tk.Canvas(root, width=300, height=200, bg="white")
canvas.pack()

canvas.create_line(10, 10, 200, 10, fill="blue", width=2)
canvas.create_rectangle(50, 50, 150, 100, fill="red")
canvas.create_oval(160, 60, 220, 120, fill="green")
canvas.create_text(150, 150, text="Canvas en Tkinter", font=("Arial", 12))

root.mainloop()
```

Lección 5: Layout y Posicionamiento en Tkinter

Tkinter proporciona tres gestores de geometría para colocar y organizar los widgets dentro de una ventana: pack(), grid() y place(). Cada uno tiene su propia lógica y ventajas, y elegir el adecuado depende del diseño deseado.

5.1 El gestor pack()

El método pack() posiciona los widgets uno tras otro (de arriba a abajo o de izquierda a derecha) en relación con su contenedor. Es útil para interfaces simples con organización vertical u horizontal.

Sintaxis:
widget.pack(opciones)

Opciones comunes:

- side: determina el lado donde se coloca el widget (TOP, BOTTOM, LEFT, RIGHT)
- fill: determina si el widget debe expandirse (NONE, X, Y, BOTH)
- expand: True o False, indica si el widget puede expandirse para ocupar espacio adicional
- padx, pady: espacio externo horizontal y vertical
- ipadx, ipady: espacio interno horizontal y vertical

Ejemplo:

```
import tkinter as tk

root = tk.Tk()
tk.Label(root, text="Arriba").pack(side="top")
tk.Label(root, text="Izquierda").pack(side="left")
tk.Label(root, text="Derecha").pack(side="right")
tk.Label(root, text="Abajo").pack(side="bottom")
root.mainloop()
```

5.2 El gestor grid()

El método grid() organiza los widgets en forma de tabla (filas y columnas). Es útil para diseños estructurados con varios elementos.

Sintaxis:
widget.grid(row=fila, column=columna, opciones)

Opciones comunes:

- rowspan, columnspan: cuántas filas o columnas ocupa el widget
- sticky: alinea el widget dentro de su celda (N, S, E, W)
- padx, pady: espacio externo horizontal y vertical
- ipadx, ipady: espacio interno horizontal y vertical

Ejemplo:

```
import tkinter as tk
```

```
root = tk.Tk()
tk.Label(root, text="Usuario:").grid(row=0, column=0, sticky="e")
tk.Entry(root).grid(row=0, column=1)
tk.Label(root, text="Contraseña:").grid(row=1, column=0, sticky="e")
tk.Entry(root, show="*").grid(row=1, column=1)
tk.Button(root, text="Entrar").grid(row=2, column=0, columnspan=2)
root.mainloop()
```

5.3 El gestor place()

El método place() posiciona widgets mediante coordenadas absolutas o relativas dentro del contenedor. Da control total pero requiere precisión.

Sintaxis:
```
widget.place(x=valor, y=valor, opciones)
```

Opciones comunes:

- x, y: coordenadas absolutas
- relx, rely: coordenadas relativas (valores entre 0 y 1)
- anchor: punto de anclaje (n, ne, e, se, s, sw, w, nw, center)

Ejemplo:
```
import tkinter as tk

root = tk.Tk()
tk.Label(root, text="Posición absoluta").place(x=50, y=30)
tk.Label(root, text="Relativo al centro").place(relx=0.5, rely=0.5, anchor="center")
root.mainloop()
```

5.4 Comparativa entre pack, grid y place

- pack() es simple y rápido, ideal para diseños lineales. Tiene limitaciones con estructuras complejas.
- grid() es ideal para formularios y diseños de tabla. Es más flexible que pack.
- place() ofrece control exacto, pero es menos adaptable y más propenso a errores en interfaces redimensionables.

Regla importante: **No mezclar grid() y pack() en el mismo contenedor**, ya que puede generar errores o comportamientos inesperados.

5.5 Uso combinado de gestores (buenas prácticas)

Aunque no se deben mezclar en un mismo contenedor, se pueden usar diferentes gestores en subcontenedores (por ejemplo, un Frame con grid dentro de un Frame con pack).

Ejemplo:
```
import tkinter as tk

root = tk.Tk()
```

```
top_frame = tk.Frame(root)
top_frame.pack()

form_frame = tk.Frame(root)
form_frame.pack()

tk.Label(top_frame, text="Encabezado").pack()

tk.Label(form_frame, text="Nombre:").grid(row=0, column=0)
tk.Entry(form_frame).grid(row=0, column=1)

tk.Label(form_frame, text="Edad:").grid(row=1, column=0)
tk.Entry(form_frame).grid(row=1, column=1)

root.mainloop()
```

5.6 Expansión y alineación de widgets

Para crear interfaces adaptables al tamaño de la ventana, se usan las siguientes estrategias:

Con pack():

- expand=True permite que el widget crezca al expandirse la ventana.
- fill="both" rellena todo el espacio disponible.

Ejemplo:

```
tk.Button(root, text="Botón adaptable").pack(fill="both", expand=True)
```

Con grid():

- sticky define la expansión dentro de la celda (por ejemplo, sticky="nsew")
- Para que funcione correctamente, se deben configurar las filas y columnas como expandibles:

```
form_frame.grid_columnconfigure(1, weight=1)
form_frame.grid_rowconfigure(0, weight=1)
```

Ejemplo:

```
import tkinter as tk

root = tk.Tk()
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)

btn = tk.Button(root, text="Expandible")
btn.grid(row=0, column=0, sticky="nsew")

root.mainloop()
```

Lección 6: Manejo de Eventos en Tkinter

Tkinter permite a los programas responder a acciones del usuario como clics, movimientos del ratón, pulsaciones de teclas, etc. Estas acciones se gestionan mediante eventos y la vinculación de estos con funciones específicas.

6.1 Vinculación de eventos (bind)

El método bind() permite asociar un evento con una función (llamada "handler") que se ejecutará cuando dicho evento ocurra sobre un widget.

Sintaxis:
widget.bind(evento, funcion)

- evento: una cadena que representa el tipo de evento (por ejemplo, "")
- funcion: la función que se ejecutará; debe aceptar un parámetro de tipo Event

Ejemplo:
import tkinter as tk

def saludo(event):
print("¡Hola! Has hecho clic.")

root = tk.Tk()
boton = tk.Button(root, text="Haz clic")
boton.pack()
boton.bind("", saludo)
root.mainloop()

6.2 Tipos de eventos (clics, teclado, ratón, etc.)

Tkinter soporta una gran variedad de eventos, algunos ejemplos comunes:

Eventos de ratón:

- : clic izquierdo
- : clic medio
- : clic derecho
- : doble clic
- : el cursor entra al widget
- : el cursor sale del widget
- : movimiento del ratón

Eventos de teclado:

- : cualquier tecla
- : cuando se presiona la tecla 'a'
- : tecla Enter
- : tecla retroceso

- : tecla ESC

Eventos de ventana:

- : se redimensiona la ventana
- , : el widget gana o pierde el foco

Ejemplo:
def mostrar_tecla(event):
print("Has presionado:", event.char)

entrada = tk.Entry(root)
entrada.pack()
entrada.bind("", mostrar_tecla)

6.3 Eventos predefinidos

Tkinter define eventos preconfigurados que pueden ser fácilmente usados con bind. Los eventos se representan con etiquetas entre ángulos, por ejemplo:

- : clic izquierdo
- : pulsación de cualquier tecla
- : movimiento del ratón
- : entrada del cursor al widget

Algunos eventos retornan datos adicionales accesibles desde el objeto `event`, como:

- event.x, event.y: coordenadas del evento
- event.char: carácter de la tecla presionada
- event.keysym: nombre simbólico de la tecla
- event.widget: widget que recibió el evento

Ejemplo:
def info_evento(event):
print("Widget:", event.widget)
print("Tecla:", event.keysym)

entrada.bind("", info_evento)

6.4 Eventos personalizados

Se pueden generar eventos personalizados en Tkinter usando el método `event_generate`.

Sintaxis:
widget.event_generate("<>")

También se pueden vincular estos eventos con funciones usando bind.

Ejemplo:
def saludo_personalizado(event):
print("¡Evento personalizado activado!")

```
root = tk.Tk()
boton = tk.Button(root, text="Disparar evento")
boton.pack()

boton.bind("<>", saludo_personalizado)

def lanzar_evento():
boton.event_generate("<>")

boton.config(command=lanzar_evento)
root.mainloop()
```

6.5 Eventos múltiples en un solo widget

Es posible vincular varios eventos distintos a un mismo widget. Cada evento puede estar asociado a una función diferente o a la misma función si se desea.

Ejemplo:

```
def click_izquierdo(event):
print("Clic izquierdo")

def click_derecho(event):
print("Clic derecho")

def doble_click(event):
print("Doble clic")

label = tk.Label(root, text="Haz clic aquí", width=30, height=5, bg="lightgray")
label.pack()

label.bind("", click_izquierdo)
label.bind("", click_derecho)
label.bind("", doble_click)
```

También se pueden combinar varios eventos en una única función usando lógica condicional basada en el contenido del evento.

Ejemplo:

```
def detectar_click(event):
if event.num == 1:
print("Clic izquierdo")
elif event.num == 3:
print("Clic derecho")

label.bind("", detectar_click)
```

Lección 7: Variables de Control en Tkinter

En Tkinter, las variables de control permiten almacenar y gestionar datos que están enlazados a widgets. Estas variables son objetos especiales que notifican automáticamente los cambios y mantienen la sincronización entre el valor del widget y el valor de la variable.

7.1 StringVar, IntVar, DoubleVar, BooleanVar

Tkinter proporciona varios tipos de variables de control, cada una diseñada para un tipo de dato específico:

- StringVar: almacena cadenas de texto
- IntVar: almacena números enteros
- DoubleVar: almacena números con decimales
- BooleanVar: almacena valores booleanos (True o False)

Ejemplo:

```
import tkinter as tk

root = tk.Tk()

texto = tk.StringVar()
entero = tk.IntVar()
decimal = tk.DoubleVar()
booleano = tk.BooleanVar()

texto.set("Hola")
entero.set(10)
decimal.set(3.14)
booleano.set(True)

print(texto.get())
print(entero.get())
print(decimal.get())
print(booleano.get())

root.mainloop()
```

7.2 Enlazar variables a widgets

Los widgets de entrada como Entry, Checkbutton, Radiobutton, Scale, Spinbox, etc., pueden enlazarse directamente con variables de control mediante el argumento `textvariable` o `variable`.

Esto permite que los cambios en el widget se reflejen automáticamente en la variable, y viceversa.

Ejemplo con Entry:

```
import tkinter as tk

root = tk.Tk()
nombre = tk.StringVar()
```

```
entrada = tk.Entry(root, textvariable=nombre)
entrada.pack()

def mostrar():
print("Nombre ingresado:", nombre.get())

boton = tk.Button(root, text="Mostrar", command=mostrar)
boton.pack()

root.mainloop()
```

Ejemplo con Checkbutton:

```
estado = tk.BooleanVar()
check = tk.Checkbutton(root, text="Aceptar", variable=estado)
check.pack()
```

Ejemplo con Radiobutton:

```
opcion = tk.StringVar()
tk.Radiobutton(root, text="Opción A", variable=opcion, value="A").pack()
tk.Radiobutton(root, text="Opción B", variable=opcion, value="B").pack()
```

7.3 Actualización automática

Una de las principales ventajas de las variables de control es que actualizan automáticamente su valor cuando el usuario interactúa con el widget, y también actualizan el widget si el valor se cambia desde el código.

Ejemplo:

```
import tkinter as tk

root = tk.Tk()
contador = tk.IntVar()

def incrementar():
contador.set(contador.get() + 1)

tk.Label(root, textvariable=contador).pack()
tk.Button(root, text="Sumar", command=incrementar).pack()

root.mainloop()
```

En este ejemplo, cada vez que se hace clic en el botón, el valor de la variable `contador` se incrementa y la etiqueta se actualiza automáticamente gracias al enlace con `textvariable`.

También se pueden usar funciones de rastreo para detectar cambios en las variables con `trace_add` (o `trace` en versiones antiguas):

Ejemplo:

```
def al_cambiar(*args):
print("Texto cambiado a:", nombre.get())

nombre = tk.StringVar()
nombre.trace_add("write", al_cambiar)
```

```
entrada = tk.Entry(root, textvariable=nombre)
entrada.pack()
```

Esto ejecuta la función `al_cambiar` cada vez que el contenido de la variable `nombre` cambia.

Lección 8: Menús y Barras en Tkinter

Los menús son una parte esencial de muchas interfaces gráficas. Tkinter permite crear menús desplegables, submenús, separadores, atajos de teclado y menús emergentes (popups) usando el widget Menu.

8.1 Menú principal (Menu)

El menú principal se crea usando el widget `Menu` y se asigna a la ventana principal con `root.config(menu=menubar)`.

Ejemplo básico:
import tkinter as tk

def accion():
print("Opción seleccionada")

root = tk.Tk()
menubar = tk.Menu(root)

menu_archivo = tk.Menu(menubar, tearoff=0)
menu_archivo.add_command(label="Nuevo", command=accion)
menu_archivo.add_command(label="Abrir", command=accion)
menu_archivo.add_command(label="Guardar", command=accion)
menu_archivo.add_separator()
menu_archivo.add_command(label="Salir", command=root.quit)

menubar.add_cascade(label="Archivo", menu=menu_archivo)

root.config(menu=menubar)
root.mainloop()

8.2 Submenús

Los submenús se crean como instancias de `Menu` y se agregan a otro menú con `add_cascade`.

Ejemplo con submenú:
menu_edicion = tk.Menu(menubar, tearoff=0)
menu_edicion.add_command(label="Copiar", command=accion)
menu_edicion.add_command(label="Pegar", command=accion)

submenu_formato = tk.Menu(menu_edicion, tearoff=0)
submenu_formato.add_command(label="Negrita")
submenu_formato.add_command(label="Cursiva")

menu_edicion.add_cascade(label="Formato", menu=submenu_formato)
menubar.add_cascade(label="Edición", menu=menu_edicion)

8.3 Separadores

Se pueden insertar separadores en un menú para agrupar visualmente opciones, usando el método `add_separator`.

Ejemplo:
menu_ayuda = tk.Menu(menubar, tearoff=0)

```
menu_ayuda.add_command(label="Documentación")
menu_ayuda.add_separator()
menu_ayuda.add_command(label="Acerca de")

menubar.add_cascade(label="Ayuda", menu=menu_ayuda)
```

8.4 Atajos de teclado en menús

Los atajos de teclado (aceleradores) se pueden mostrar visualmente en los menús con el argumento `accelerator`, pero para que funcionen, deben vincularse manualmente usando `bind`.

Ejemplo:

```
def guardar():
print("Archivo guardado")

root.bind("", lambda event: guardar())

menu_archivo.add_command(label="Guardar", command=guardar, accelerator="Ctrl+S")
```

Nota: Los aceleradores no se ejecutan automáticamente, deben vincularse mediante bind a un evento de teclado.

8.5 Menús emergentes (popup)

Los menús emergentes aparecen al hacer clic derecho. Se crean como menús normales pero se muestran usando `post(x, y)`.

Ejemplo:

```
def opcion_popup():
print("Opción del menú emergente")

popup = tk.Menu(root, tearoff=0)
popup.add_command(label="Opción 1", command=opcion_popup)
popup.add_command(label="Opción 2", command=opcion_popup)

def mostrar_popup(event):
popup.post(event.x_root, event.y_root)

root.bind("", mostrar_popup)
```

En sistemas Linux es `<Button-3>` (clic derecho). En macOS puede necesitarse usar otro botón según configuración del ratón.

Lección 9: Cuadros de Diálogo en Tkinter

Tkinter incluye varios módulos que permiten mostrar cuadros de diálogo estándar para interactuar con el usuario. Estos cuadros de diálogo son útiles para mostrar mensajes, abrir y guardar archivos, elegir colores y recibir entradas simples.

9.1 messagebox: información, advertencia, error

El módulo `tkinter.messagebox` permite mostrar ventanas emergentes con mensajes al usuario. Los cuadros disponibles incluyen información, advertencia, error, confirmación, etc.

Importación:
from tkinter import messagebox

Ejemplos:
messagebox.showinfo("Información", "Operación completada con éxito")
messagebox.showwarning("Advertencia", "Estás por eliminar un archivo")
messagebox.showerror("Error", "No se pudo abrir el archivo")

También hay cuadros con botones que devuelven un valor:
respuesta = messagebox.askquestion("¿Deseas continuar?", "Esta acción no se puede deshacer")
if respuesta == "yes":
print("Usuario eligió continuar")

Otros métodos:

- askyesno() → devuelve True o False
- askokcancel() → devuelve True o False
- askretrycancel() → devuelve True o False

9.2 filedialog: abrir y guardar archivos

El módulo `tkinter.filedialog` permite abrir cuadros de diálogo para seleccionar archivos o directorios.

Importación:
from tkinter import filedialog

Abrir archivo:
archivo = filedialog.askopenfilename(title="Selecciona un archivo", filetypes=[("Archivos de texto", "_.txt"), ("Todos los archivos", "_.*")])
print("Archivo seleccionado:", archivo)

Guardar archivo:
archivo = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Archivos de texto", "*.txt")])
print("Guardar como:", archivo)

Seleccionar directorio:
directorio = filedialog.askdirectory()
print("Directorio seleccionado:", directorio)

9.3 colorchooser: seleccionar color

El módulo `tkinter.colorchooser` permite al usuario elegir un color mediante un cuadro de selección de color.

Importación:
from tkinter import colorchooser

color = colorchooser.askcolor(title="Elige un color")
print("Color RGB:", color[0])
print("Código hexadecimal:", color[1])

El resultado es una tupla (RGB, HEX), por ejemplo: ((255, 0, 0), "#ff0000")

9.4 simpledialog: entradas simples del usuario

El módulo `tkinter.simpledialog` permite solicitar entradas al usuario como texto, números enteros o flotantes.

Importación:
from tkinter import simpledialog

Solicitar una cadena:
nombre = simpledialog.askstring("Entrada", "¿Cuál es tu nombre?")
print("Nombre:", nombre)

Solicitar un número entero:
edad = simpledialog.askinteger("Entrada", "¿Cuántos años tienes?")
print("Edad:", edad)

Solicitar un número decimal:
altura = simpledialog.askfloat("Entrada", "¿Cuál es tu altura en metros?")
print("Altura:", altura)

Estos cuadros devuelven el valor ingresado por el usuario o `None` si se canceló.

Lección 10: Canvas y Gráficos en Tkinter

El widget Canvas permite dibujar gráficos, formas, texto e imágenes dentro de una ventana Tkinter. Es muy útil para crear interfaces gráficas personalizadas, juegos simples o visualizaciones.

10.1 Crear un lienzo (Canvas)

Para crear un Canvas se usa `tk.Canvas` y se coloca dentro de una ventana o contenedor.

Ejemplo básico:
import tkinter as tk

root = tk.Tk()
canvas = tk.Canvas(root, width=400, height=300, bg="white")
canvas.pack()

root.mainloop()

10.2 Dibujo de líneas, óvalos, rectángulos, polígonos, texto

El Canvas ofrece varios métodos para dibujar:

- `create_line(x1, y1, x2, y2, ...)` dibuja una línea entre puntos.
- `create_oval(x1, y1, x2, y2)` dibuja un óvalo dentro del rectángulo definido por las coordenadas.
- `create_rectangle(x1, y1, x2, y2)` dibuja un rectángulo.
- `create_polygon(x1, y1, x2, y2, x3, y3, ..., options)` dibuja un polígono con los puntos indicados.
- `create_text(x, y, text="texto")` dibuja texto en la posición indicada.

Ejemplo:
canvas.create_line(10, 10, 200, 10, fill="blue", width=2)
canvas.create_oval(50, 50, 150, 100, outline="red", fill="yellow")
canvas.create_rectangle(200, 50, 300, 150, outline="green", fill="lightgreen")
canvas.create_polygon(300, 200, 350, 250, 250, 250, fill="purple")
canvas.create_text(200, 200, text="Hola Canvas", font=("Arial", 16), fill="black")

10.3 Movimiento de objetos

Los objetos dibujados en Canvas pueden moverse usando el método `move` que recibe el ID del objeto y los desplazamientos en x e y.

Ejemplo:
id_linea = canvas.create_line(10, 10, 200, 10, fill="blue", width=2)

def mover_derecha():
canvas.move(id_linea, 10, 0)

boton = tk.Button(root, text="Mover línea", command=mover_derecha)
boton.pack()

10.4 Detección de colisiones y coordenadas

Se pueden obtener las coordenadas de un objeto con `coords(id_objeto)` y detectar colisiones o superposiciones analizando estas coordenadas.

También se puede usar `find_overlapping(x1, y1, x2, y2)` para encontrar objetos que se superponen con un área.

Ejemplo:
coords = canvas.coords(id_linea)
print("Coordenadas:", coords)

superpuestos = canvas.find_overlapping(0, 0, 100, 100)
print("Objetos en área:", superpuestos)

10.5 Imágenes en el lienzo

El Canvas puede mostrar imágenes usando `create_image(x, y, image=imagen_objeto)`.

Las imágenes deben ser objetos `PhotoImage` (de Tkinter) o compatibles.

Ejemplo:
imagen = tk.PhotoImage(file="imagen.png")
canvas.create_image(100, 100, image=imagen, anchor=tk.NW)

Nota: La referencia a la imagen debe mantenerse para que no sea eliminada por el recolector de basura (guardar la variable en ámbito global o atributo).

:Lección 11: Trabajar con Imágenes en Tkinter

Tkinter permite trabajar con imágenes para enriquecer las interfaces gráficas, ya sea para mostrar iconos, fotos o gráficos. La gestión básica se hace con PhotoImage, y para formatos avanzados se utiliza PIL (Pillow).

11.1 Cargar imágenes con PhotoImage

El widget PhotoImage carga imágenes en formatos compatibles (normalmente GIF, PNG, PGM, PPM). Se crea un objeto PhotoImage que luego se puede asignar a widgets como Label, Button, Canvas, etc.

Ejemplo:

```
import tkinter as tk

root = tk.Tk()
imagen = tk.PhotoImage(file="ejemplo.png")
label = tk.Label(root, image=imagen)
label.pack()

root.mainloop()
```

Nota: La variable que contiene la imagen debe mantenerse viva mientras se usa (no debe ser variable local que desaparezca).

11.2 Formatos compatibles

Por defecto, PhotoImage soporta:

- PNG
- GIF
- PGM
- PPM

No soporta formatos comunes como JPEG o BMP, para eso es necesario usar Pillow.

11.3 Mostrar imágenes en widgets

Además de Label, otros widgets que pueden mostrar imágenes:

- Button: puede mostrar imagen junto al texto o solo la imagen.
- Canvas: con `create_image`.
- Checkbutton, Radiobutton: pueden mostrar imágenes en lugar o junto al texto.

Ejemplo en Button:

```
btn = tk.Button(root, image=imagen)
btn.pack()
```

11.4 Uso de PIL (Pillow) para formatos avanzados

Pillow es una biblioteca externa que permite trabajar con muchos formatos de imagen, incluyendo JPEG, BMP, TIFF y más.

Para usarla con Tkinter:

1. Instalar Pillow (fuera de Tkinter): `pip install pillow`
2. Importar módulos:
   from PIL import Image, ImageTk
3. Cargar imagen:
   imagen_pil = Image.open("foto.jpg")
4. Convertir para Tkinter:
   imagen_tk = ImageTk.PhotoImage(imagen_pil)
5. Usar en widget:
   label = tk.Label(root, image=imagen_tk)
   label.image = imagen_tk # mantener referencia
   label.pack()

Pillow también permite manipular imágenes: redimensionar, recortar, aplicar filtros, etc.

Lección 12. Widgets Avanzados (ttk)

12.1 Introducción a ttk (Themed Tk)

Tkinter tiene un módulo llamado `ttk` (Themed Tkinter) que ofrece widgets con un diseño más moderno y estilizado que los widgets básicos de Tkinter. Estos widgets utilizan temas nativos del sistema operativo y permiten la personalización a través de estilos (`Style`). Para usar los widgets ttk, se debe importar de la siguiente manera:

```
from tkinter import ttk
```

Los widgets de `ttk` reemplazan a sus equivalentes clásicos de Tkinter, y también incluyen algunos nuevos como `Combobox`, `Notebook`, `Treeview`, `Progressbar`, etc.

12.2 ttk.Label, ttk.Entry, ttk.Button, ttk.Checkbutton

Los widgets básicos de ttk se comportan igual que sus contrapartes en Tkinter, pero con mejor apariencia.

Ejemplo:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Ejemplo ttk básicos")

ttk.Label(ventana, text="Nombre:").grid(row=0, column=0)
ttk.Entry(ventana).grid(row=0, column=1)

ttk.Checkbutton(ventana, text="Acepto").grid(row=1, columnspan=2)

ttk.Button(ventana, text="Enviar").grid(row=2, columnspan=2)

ventana.mainloop()
```

12.3 Combobox

El widget `ttk.Combobox` permite seleccionar un valor de una lista desplegable.

Ejemplo:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Ejemplo Combobox")

valores = ["Python", "Java", "C++", "C#"]
combo = ttk.Combobox(ventana, values=valores)
combo.current(0)
combo.pack()

ventana.mainloop()
```

12.4 Notebook (pestañas)

El widget `ttk.Notebook` permite crear interfaces con pestañas, útiles para organizar contenidos por secciones.

Ejemplo:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Ejemplo Notebook")

notebook = ttk.Notebook(ventana)

tab1 = ttk.Frame(notebook)
tab2 = ttk.Frame(notebook)

notebook.add(tab1, text="Pestaña 1")
notebook.add(tab2, text="Pestaña 2")

notebook.pack(expand=True, fill='both')

ttk.Label(tab1, text="Contenido de la pestaña 1").pack(pady=10)
ttk.Label(tab2, text="Contenido de la pestaña 2").pack(pady=10)

ventana.mainloop()
```

12.5 Treeview (listas y jerarquías)

`ttk.Treeview` se usa para mostrar datos jerárquicos o tabulares, similar a una tabla con columnas y filas.

Ejemplo:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Ejemplo Treeview")

tree = ttk.Treeview(ventana, columns=("Edad", "País"), show="headings")
tree.heading("Edad", text="Edad")
tree.heading("País", text="País")

tree.insert("", tk.END, values=("25", "España"))
tree.insert("", tk.END, values=("30", "México"))

tree.pack()

ventana.mainloop()
```

12.6 Progressbar

El widget `ttk.Progressbar` sirve para mostrar el progreso de una operación. Puede estar en modo indeterminado o determinado.

Ejemplo (modo indeterminado):

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Ejemplo Progressbar")

barra = ttk.Progressbar(ventana, mode="indeterminate")
barra.pack()
```

```
barra.start()

ventana.mainloop()
```

Ejemplo (modo determinado):

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Progressbar Determinada")

barra = ttk.Progressbar(ventana, maximum=100, length=200)
barra.pack()
barra["value"] = 70

ventana.mainloop()
```

12.7 Separator

`ttk.Separator` se utiliza para dividir visualmente secciones de una interfaz.

Ejemplo:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Ejemplo Separator")

ttk.Label(ventana, text="Encabezado").pack()

ttk.Separator(ventana, orient="horizontal").pack(fill='x', pady=10)

ttk.Label(ventana, text="Contenido").pack()

ventana.mainloop()
```

12.8 Style: personalización de temas y estilos

`ttk.Style` permite modificar los estilos visuales de los widgets ttk o crear nuevos temas. Se puede cambiar el tema global o modificar estilos de widgets específicos.

Ejemplo de cambiar tema:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Estilo ttk")

style = ttk.Style()
style.theme_use("clam")

ttk.Button(ventana, text="Botón estilizado").pack(pady=10)

ventana.mainloop()
```

Ejemplo de personalización:

```
import tkinter as tk
from tkinter import ttk
```

```
ventana = tk.Tk()
ventana.title("Personalizar Estilo")

style = ttk.Style()
style.configure("Custom.TButton", foreground="blue", background="lightgray",
font=("Arial", 12))

ttk.Button(ventana, text="Botón Personalizado",
style="Custom.TButton").pack(pady=10)

ventana.mainloop()
```

Lección 13. Validación de Entradas en Tkinter

13.1 Métodos de validación (validate, validatecommand)

Tkinter permite validar entradas en widgets `Entry` utilizando dos opciones clave: `validate` y `validatecommand`.
La opción `validate` define cuándo se ejecutará la validación (`'focusin'`, `'focusout'`, `'key'`, `'all'`, etc.).
La opción `validatecommand` toma una función que será llamada cada vez que se dispare el evento de validación.

Para poder usar `validatecommand`, hay que registrar primero la función en el método `register`.

Ejemplo básico de validación:

```
import tkinter as tk

def validar_entrada(texto):
    return texto.isdigit()

ventana = tk.Tk()

vcmd = ventana.register(validar_entrada)

entrada = tk.Entry(ventana, validate='key', validatecommand=(vcmd, '%P'))
entrada.pack()

ventana.mainloop()
```

En este ejemplo, la entrada solo aceptará números. `%P` es un código de sustitución que representa el nuevo contenido del `Entry`.

13.2 Restricción de datos (números, letras, longitudes)

Para restringir los datos que puede ingresar el usuario, podemos usar funciones de validación que verifiquen:

- Si el texto es solo numérico o alfabético
- Si tiene una longitud determinada

Ejemplo para aceptar solo letras y limitar a 10 caracteres:

```
import tkinter as tk

def validar_letras(texto):
    return texto.isalpha() and len(texto) <= 10

ventana = tk.Tk()
vcmd = ventana.register(validar_letras)

entrada = tk.Entry(ventana, validate='key', validatecommand=(vcmd, '%P'))
entrada.pack()

ventana.mainloop()
```

Ejemplo para permitir solo números entre 1 y 100:

```
import tkinter as tk

def validar_rango(texto):
    if texto == '':
        return True
    if texto.isdigit():
        valor = int(texto)
        return 1 <= valor <= 100
    return False

ventana = tk.Tk()
vcmd = ventana.register(validar_rango)

entrada = tk.Entry(ventana, validate='key', validatecommand=(vcmd, '%P'))
entrada.pack()

ventana.mainloop()
```

13.3 Validación con expresiones regulares

Python permite usar expresiones regulares para validar patrones más complejos. Usamos el módulo `re`.

Ejemplo para permitir un correo electrónico válido:

```
import tkinter as tk
import re

def validar_email(texto):
    patron = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    return re.fullmatch(patron, texto) is not None

ventana = tk.Tk()
vcmd = ventana.register(validar_email)

entrada = tk.Entry(ventana, validate='focusout', validatecommand=(vcmd, '%P'))
entrada.pack()

ventana.mainloop()
```

En este ejemplo, la validación se realiza cuando el foco sale del campo (`focusout`). Solo se aceptarán valores que coincidan con el patrón de un correo electrónico.

Notas adicionales:

- `%P` es el nuevo valor propuesto para el campo.
- Otros códigos útiles son: `%S` (último carácter introducido), `%V` (tipo de validación), `%W` (nombre del widget).
- Siempre se recomienda hacer pruebas con entradas vacías si se permite borrar completamente el campo.

Resumen:

- `validate` define cuándo validar.
- `validatecommand` llama a una función validadora.

- Podemos combinar lógica booleana y expresiones regulares para validar tipos de entrada, rangos, patrones, etc.

Esta lección completa permite construir formularios robustos en Tkinter con validaciones específicas y seguras.

Lección 14. Temporizadores y Animaciones en Tkinter

14.1 Uso de after y after_cancel

Tkinter ofrece el método after para ejecutar una función después de un cierto tiempo en milisegundos. También permite usar after_cancel para cancelar una llamada programada con after.

Sintaxis básica:
widget.after(tiempo_en_ms, funcion, argumentos)
widget.after_cancel(id)

Ejemplo simple:
import tkinter as tk

def mostrar_mensaje():
etiqueta.config(text="Hola después de 2 segundos")

ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="Esperando...")
etiqueta.pack()

ventana.after(2000, mostrar_mensaje)

ventana.mainloop()

El mensaje cambia después de 2000 milisegundos. El método after devuelve un identificador que puede ser usado para cancelar la llamada con after_cancel.

Ejemplo con cancelación:
import tkinter as tk

def cambiar_mensaje():
etiqueta.config(text="Cambio programado")

def cancelar():
ventana.after_cancel(id_temporizador)

ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="Esperando...")
etiqueta.pack()

id_temporizador = ventana.after(5000, cambiar_mensaje)

boton_cancelar = tk.Button(ventana, text="Cancelar", command=cancelar)
boton_cancelar.pack()

ventana.mainloop()

14.2 Temporizadores con funciones periódicas

Se puede usar after de forma repetida para ejecutar una función periódicamente, como si fuera un temporizador o bucle.

Ejemplo de reloj que se actualiza cada segundo:
import tkinter as tk
import time

```
def actualizar_reloj():
hora = time.strftime("%H:%M:%S")
etiqueta.config(text=hora)
ventana.after(1000, actualizar_reloj)

ventana = tk.Tk()
etiqueta = tk.Label(ventana, font=("Helvetica", 30))
etiqueta.pack()

actualizar_reloj()

ventana.mainloop()
```

La función se llama a sí misma cada 1000 milisegundos, creando una ejecución continua.

14.3 Crear animaciones básicas en canvas

Con el método after y el widget Canvas es posible mover objetos para crear animaciones simples.

Ejemplo de un círculo que se mueve a la derecha:

```
import tkinter as tk

def mover():
canvas.move(circulo, 5, 0)
canvas.after(50, mover)

ventana = tk.Tk()
canvas = tk.Canvas(ventana, width=400, height=200)
canvas.pack()

circulo = canvas.create_oval(10, 50, 60, 100, fill="blue")

mover()

ventana.mainloop()
```

En este ejemplo, cada 50 milisegundos se mueve el círculo 5 píxeles hacia la derecha. Esto crea una animación suave y continua.

14.4 Repetición automática de acciones

Cualquier acción, como cambiar colores, mover elementos o actualizar datos, puede repetirse automáticamente usando after.

Ejemplo de cambio de color alternado:

```
import tkinter as tk

def cambiar_color():
color_actual = etiqueta.cget("bg")
nuevo_color = "red" if color_actual == "blue" else "blue"
etiqueta.config(bg=nuevo_color)
ventana.after(1000, cambiar_color)

ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="Parpadeando", bg="blue", fg="white", font=("Arial", 24))
etiqueta.pack(pady=20)
```

cambiar_color()

ventana.mainloop()

Este ejemplo cambia el color de fondo del texto automáticamente cada segundo. La función se llama a sí misma indefinidamente.

Resumen:

- after permite ejecutar funciones después de cierto tiempo.
- after_cancel puede cancelar una llamada programada.
- El uso repetido de after permite temporizadores periódicos.
- Se pueden crear animaciones y repeticiones automáticas.
- El método es útil para interfaces dinámicas, juegos y relojes.

Esta lección cubre el control del tiempo y la creación de efectos visuales simples en interfaces gráficas con Tkinter.

Lección 15. Hilos (Threads) y Concurrencia en Tkinter

15.1 Uso de threading en interfaces

Tkinter corre sobre un solo hilo principal. Si se ejecutan operaciones largas directamente en funciones de botones u otros eventos, la interfaz se congela. Para evitarlo, se puede usar el módulo threading de Python y ejecutar tareas pesadas en un hilo separado.

Ejemplo básico con threading:

```
import tkinter as tk
import threading
import time

def tarea_larga():
time.sleep(5)
print("Tarea terminada")

def iniciar_tarea():
hilo = threading.Thread(target=tarea_larga)
hilo.start()

ventana = tk.Tk()
boton = tk.Button(ventana, text="Iniciar", command=iniciar_tarea)
boton.pack()

ventana.mainloop()
```

En este ejemplo, la tarea de 5 segundos se ejecuta en un hilo aparte, permitiendo que la ventana permanezca activa.

15.2 Evitar bloqueos en la interfaz

Al usar tareas largas sin hilos, la interfaz gráfica deja de responder. Esto ocurre porque el hilo principal de Tkinter se bloquea esperando a que termine la función.

Ejemplo del problema (no recomendado):

```
def tarea_larga():
time.sleep(5)
etiqueta.config(text="Hecho")

boton = tk.Button(ventana, text="Iniciar", command=tarea_larga)
```

Este código hace que la interfaz se congele por 5 segundos. Para evitarlo, se debe usar threading como en el ejemplo anterior.

Ejemplo mejorado con indicador visual:

```
def tarea_larga():
time.sleep(5)
ventana.after(0, lambda: etiqueta.config(text="Hecho"))

def iniciar_tarea():
etiqueta.config(text="Procesando...")
```

```
hilo = threading.Thread(target=tarea_larga)
hilo.start()

boton = tk.Button(ventana, text="Iniciar", command=iniciar_tarea)
```

15.3 Comunicación entre hilos y Tkinter

Tkinter no es seguro para acceso directo desde otros hilos. Por lo tanto, para actualizar la interfaz desde un hilo secundario, se debe usar el método after del hilo principal.

Ejemplo con comunicación segura entre hilos:

```
import tkinter as tk
import threading
import time

def tarea_larga():
time.sleep(5)
ventana.after(0, actualizar_interfaz)

def actualizar_interfaz():
etiqueta.config(text="Tarea completada")

def iniciar_tarea():
etiqueta.config(text="Trabajando...")
hilo = threading.Thread(target=tarea_larga)
hilo.start()

ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="Listo")
etiqueta.pack()

boton = tk.Button(ventana, text="Iniciar tarea", command=iniciar_tarea)
boton.pack()

ventana.mainloop()
```

En este ejemplo, la función tarea_larga se ejecuta en segundo plano y, una vez completada, llama a una función del hilo principal usando after para modificar la interfaz de forma segura.

Resumen:

- Tkinter se ejecuta en un solo hilo principal.
- Las tareas largas deben colocarse en un hilo separado con threading para evitar bloqueos.
- No se debe modificar la interfaz directamente desde hilos secundarios.
- Se debe usar after para que los hilos secundarios soliciten actualizaciones de forma segura al hilo principal.
- Este enfoque mejora la experiencia del usuario al mantener la interfaz receptiva.

Esta lección permite crear interfaces más robustas y reactivas que manejan tareas en segundo plano sin afectar la experiencia visual o el control de la aplicación.

Lección 16. Internacionalización en Tkinter

16.1 Mostrar diferentes idiomas

La internacionalización permite que una aplicación se adapte a distintos idiomas sin modificar su código principal. En Tkinter, esto se logra cargando textos traducidos desde archivos o estructuras de datos, y aplicándolos a los widgets.

Ejemplo simple con soporte para inglés y español:

```
import tkinter as tk

idiomas = {
"es": {
"saludo": "Hola",
"boton": "Aceptar"
},
"en": {
"saludo": "Hello",
"boton": "Accept"
}
}

idioma_actual = "es"

def cambiar_idioma(nuevo_idioma):
global idioma_actual
idioma_actual = nuevo_idioma
etiqueta.config(text=idiomas[idioma_actual]["saludo"])
boton.config(text=idiomas[idioma_actual]["boton"])

ventana = tk.Tk()

etiqueta = tk.Label(ventana, text=idiomas[idioma_actual]["saludo"])
etiqueta.pack()

boton = tk.Button(ventana, text=idiomas[idioma_actual]["boton"])
boton.pack()

tk.Button(ventana, text="ES", command=lambda: cambiar_idioma("es")).pack(side="left")
tk.Button(ventana, text="EN", command=lambda: cambiar_idioma("en")).pack(side="right")

ventana.mainloop()
```

Este ejemplo cambia dinámicamente los textos según el idioma seleccionado.

16.2 Codificación de texto y fuentes

Tkinter soporta caracteres Unicode, por lo que puede mostrar texto en muchos idiomas siempre que los archivos estén guardados con codificación UTF-8 y se use una fuente compatible.

Ejemplo con texto en varios idiomas:

```
import tkinter as tk
```

```
ventana = tk.Tk()

texto = "Español: Hola\nEnglish: Hello\nРусский: Привет\n 中文: 你好\nالعربية: مرحبا"

etiqueta = tk.Label(ventana, text=texto, font=("Arial", 14))
etiqueta.pack(padx=20, pady=20)

ventana.mainloop()
```

Es importante usar fuentes que soporten los caracteres especiales del idioma deseado, como Arial, Noto Sans o DejaVu.

Para evitar errores de codificación, se debe guardar el archivo fuente como UTF-8 y evitar codificaciones obsoletas como Latin-1.

16.3 Archivos de traducción

Una práctica común para manejar traducciones es usar archivos externos. Uno de los métodos más usados en Python es el módulo gettext, que permite mantener archivos de traducción separados por idioma.

Ejemplo básico con gettext:

1. Crear archivo de texto traducciones.po:

```
msgid "saludo"
msgstr "Hola"

msgid "boton"
msgstr "Aceptar"
```

2. Compilarlo a archivo .mo y usarlo en el programa:

```
import tkinter as tk
import gettext
import os

idioma = "es"
localedir = os.path.join(os.path.dirname(file), "locales")
lang = gettext.translation("mensajes", localedir=localedir, languages=[idioma])
lang.install()
_ = lang.gettext

ventana = tk.Tk()

etiqueta = tk.Label(ventana, text=_("saludo"))
etiqueta.pack()

boton = tk.Button(ventana, text=_("boton"))
boton.pack()

ventana.mainloop()
```

Este método permite mantener separados los textos traducibles del código y facilita el mantenimiento del programa en varios idiomas.

Resumen:

- La internacionalización permite adaptar la interfaz a diferentes idiomas.
- Se pueden usar diccionarios en memoria para cargar traducciones básicas.
- Para soportar múltiples idiomas con escalabilidad, se recomienda usar gettext y archivos .po/.mo.
- Se debe guardar el archivo fuente en UTF-8 y usar fuentes que soporten caracteres multilingües.
- La internacionalización mejora la accesibilidad global de las aplicaciones hechas en Tkinter.

Lección 17. Guardar y Cargar Datos en Tkinter

17.1 Uso de archivos .txt, .csv, .json

Tkinter permite integrar fácilmente el guardado y la carga de datos mediante archivos de texto planos como .txt, .csv y .json. Esto permite almacenar datos de formularios, configuraciones o historiales de usuario.

Guardar y cargar un archivo .txt:

```
import tkinter as tk
from tkinter import filedialog

def guardar_txt():
texto = entrada.get()
archivo = filedialog.asksaveasfilename(defaultextension=".txt")
if archivo:
with open(archivo, "w", encoding="utf-8") as f:
f.write(texto)

def cargar_txt():
archivo = filedialog.askopenfilename(filetypes=[("Archivos de texto", "*.txt")])
if archivo:
with open(archivo, "r", encoding="utf-8") as f:
contenido = f.read()
entrada.delete(0, tk.END)
entrada.insert(0, contenido)

ventana = tk.Tk()
entrada = tk.Entry(ventana, width=40)
entrada.pack()

tk.Button(ventana, text="Guardar", command=guardar_txt).pack()
tk.Button(ventana, text="Cargar", command=cargar_txt).pack()

ventana.mainloop()
```

Guardar y cargar archivo .json:

```
import json

def guardar_json():
datos = {"nombre": entrada.get()}
with open("datos.json", "w", encoding="utf-8") as f:
json.dump(datos, f)

def cargar_json():
with open("datos.json", "r", encoding="utf-8") as f:
datos = json.load(f)
entrada.delete(0, tk.END)
entrada.insert(0, datos["nombre"])
```

Guardar y cargar archivo .csv:

```
import csv

def guardar_csv():
with open("datos.csv", "w", newline="", encoding="utf-8") as f:
escritor = csv.writer(f)
escritor.writerow(["Nombre"])
escritor.writerow([entrada.get()])

def cargar_csv():
with open("datos.csv", "r", encoding="utf-8") as f:
lector = csv.reader(f)
next(lector)
fila = next(lector)
entrada.delete(0, tk.END)
entrada.insert(0, fila[0])
```

17.2 Serialización con pickle

El módulo pickle permite guardar y recuperar estructuras de datos complejas como listas, diccionarios u objetos Python.

Ejemplo:

```
import pickle

def guardar_pickle():
datos = {"nombre": entrada.get()}
with open("datos.pkl", "wb") as f:
pickle.dump(datos, f)

def cargar_pickle():
with open("datos.pkl", "rb") as f:
datos = pickle.load(f)
entrada.delete(0, tk.END)
entrada.insert(0, datos["nombre"])
```

Pickle guarda los datos en formato binario, por lo que no son legibles directamente, pero permiten guardar estructuras Python completas fácilmente.

17.3 Integración con bases de datos (SQLite)

SQLite es una base de datos integrada en Python que no necesita instalación adicional. Es útil para guardar datos estructurados de forma persistente.

Ejemplo básico de insertar y recuperar datos:

```
import sqlite3

def crear_tabla():
conexion = sqlite3.connect("usuarios.db")
cursor = conexion.cursor()
cursor.execute("CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY,
nombre TEXT)")
```

```
conexion.commit()
conexion.close()

def guardar_sqlite():
nombre = entrada.get()
conexion = sqlite3.connect("usuarios.db")
cursor = conexion.cursor()
cursor.execute("INSERT INTO usuarios (nombre) VALUES (?)", (nombre,))
conexion.commit()
conexion.close()

def cargar_sqlite():
conexion = sqlite3.connect("usuarios.db")
cursor = conexion.cursor()
cursor.execute("SELECT nombre FROM usuarios ORDER BY id DESC LIMIT 1")
fila = cursor.fetchone()
if fila:
entrada.delete(0, tk.END)
entrada.insert(0, fila[0])
conexion.close()

crear_tabla()

ventana = tk.Tk()
entrada = tk.Entry(ventana, width=40)
entrada.pack()

tk.Button(ventana, text="Guardar SQL", command=guardar_sqlite).pack()
tk.Button(ventana, text="Cargar SQL", command=cargar_sqlite).pack()

ventana.mainloop()
```

Resumen:

- Se puede usar .txt, .csv y .json para guardar datos estructurados simples.
- Pickle permite guardar estructuras Python completas en formato binario.
- SQLite proporciona una base de datos integrada ideal para guardar datos organizados a largo plazo.
- Tkinter se puede combinar fácilmente con estos formatos para crear formularios, registros y configuraciones persistentes.
- El uso de interfaces de archivos o bases de datos permite construir aplicaciones funcionales y escalables.

Lección 18. Ejemplos Prácticos en Tkinter

18.1 Calculadora

Una calculadora básica con botones numéricos y de operaciones:

```
import tkinter as tk

def agregar(valor):
    entrada.insert(tk.END, valor)

def calcular():
    try:
        resultado = eval(entrada.get())
        entrada.delete(0, tk.END)
        entrada.insert(tk.END, str(resultado))
    except:
        entrada.delete(0, tk.END)
        entrada.insert(tk.END, "Error")

def limpiar():
    entrada.delete(0, tk.END)

ventana = tk.Tk()
entrada = tk.Entry(ventana, width=20, font=("Arial", 18))
entrada.grid(row=0, column=0, columnspan=4)

botones = [
    "7", "8", "9", "+",
    "4", "5", "6", "-",
    "1", "2", "3", "*",
    "0", ".", "=", "/"
]

fila = 1
col = 0
for b in botones:
    accion = lambda x=b: calcular() if x == "=" else agregar(x)
    tk.Button(ventana, text=b, width=5, height=2, command=accion).grid(row=fila, 
column=col)
    col += 1
    if col > 3:
        col = 0
        fila += 1

tk.Button(ventana, text="C", command=limpiar, width=20).grid(row=5, column=0, 
columnspan=4)

ventana.mainloop()
```

18.2 Reloj digital

```
import tkinter as tk
import time

def actualizar():
    hora = time.strftime("%H:%M:%S")
    etiqueta.config(text=hora)
    ventana.after(1000, actualizar)

ventana = tk.Tk()
etiqueta = tk.Label(ventana, font=("Arial", 40))
etiqueta.pack()
```

```
actualizar()
ventana.mainloop()
```

18.3 Editor de texto

```
from tkinter import *
from tkinter import filedialog

def abrir():
    archivo = filedialog.askopenfilename()
    if archivo:
        with open(archivo, "r", encoding="utf-8") as f:
            texto.delete("1.0", END)
            texto.insert(END, f.read())

def guardar():
    archivo = filedialog.asksaveasfilename(defaultextension=".txt")
    if archivo:
        with open(archivo, "w", encoding="utf-8") as f:
            f.write(texto.get("1.0", END))

ventana = Tk()
ventana.title("Editor de texto")
texto = Text(ventana)
texto.pack(expand=1, fill=BOTH)

menu = Menu(ventana)
ventana.config(menu=menu)
archivo_menu = Menu(menu, tearoff=0)
menu.add_cascade(label="Archivo", menu=archivo_menu)
archivo_menu.add_command(label="Abrir", command=abrir)
archivo_menu.add_command(label="Guardar", command=guardar)
ventana.mainloop()
```

18.4 Agenda de contactos

```
import tkinter as tk

contactos = []

def agregar():
    nombre = entrada.get()
    if nombre:
        lista.insert(tk.END, nombre)
        contactos.append(nombre)
        entrada.delete(0, tk.END)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Agregar", command=agregar).pack()
lista = tk.Listbox(ventana)
lista.pack()
ventana.mainloop()
```

18.5 Visor de imágenes

```
from tkinter import *
from tkinter import filedialog
from PIL import ImageTk, Image

def cargar():
```

```
    ruta = filedialog.askopenfilename(filetypes=[("Imágenes", "*.jpg *.png
*.gif")])
    if ruta:
        imagen = Image.open(ruta)
        imagen = imagen.resize((300, 300))
        foto = ImageTk.PhotoImage(imagen)
        etiqueta.config(image=foto)
        etiqueta.image = foto

ventana = Tk()
tk.Button(ventana, text="Cargar imagen", command=cargar).pack()
etiqueta = tk.Label(ventana)
etiqueta.pack()
ventana.mainloop()
```

18.6 Reproductor de audio simple

```
import tkinter as tk
from tkinter import filedialog
import pygame

pygame.init()

def reproducir():
    archivo = filedialog.askopenfilename(filetypes=[("Audio", "*.mp3")])
    if archivo:
        pygame.mixer.music.load(archivo)
        pygame.mixer.music.play()

def detener():
    pygame.mixer.music.stop()

ventana = tk.Tk()
tk.Button(ventana, text="Reproducir", command=reproducir).pack()
tk.Button(ventana, text="Detener", command=detener).pack()
ventana.mainloop()
```

18.7 Juego de memoria

```
import tkinter as tk
import random

pares = ["A", "A", "B", "B", "C", "C", "D", "D"]
random.shuffle(pares)
botones = []
seleccionados = []

def revelar(i):
    if len(seleccionados) < 2 and botones[i]["text"] == "":
        botones[i]["text"] = pares[i]
        seleccionados.append(i)
        if len(seleccionados) == 2:
            ventana.after(1000, comprobar)

def comprobar():
    i, j = seleccionados
    if pares[i] != pares[j]:
        botones[i]["text"] = ""
        botones[j]["text"] = ""
    seleccionados.clear()

ventana = tk.Tk()
for i in range(8):
```

```
    b = tk.Button(ventana, text="", width=5, height=2, command=lambda i=i:
revelar(i))
    b.grid(row=i//4, column=i%4)
    botones.append(b)
ventana.mainloop()
```

18.8 Chat simple en red

Servidor:

```
import socket
import threading

clientes = []

def manejar(cliente):
    while True:
        try:
            mensaje = cliente.recv(1024)
            for c in clientes:
                if c != cliente:
                    c.send(mensaje)
        except:
            clientes.remove(cliente)
            cliente.close()
            break

servidor = socket.socket()
servidor.bind(("localhost", 5000))
servidor.listen(5)
print("Servidor iniciado")

while True:
    cliente, _ = servidor.accept()
    clientes.append(cliente)
    threading.Thread(target=manejar, args=(cliente,)).start()
```

Cliente con Tkinter:

```
import tkinter as tk
import socket
import threading

def recibir():
    while True:
        try:
            msg = cliente.recv(1024).decode()
            chat.insert(tk.END, msg + "\n")
        except:
            break

def enviar():
    msg = entrada.get()
    cliente.send(msg.encode())
    entrada.delete(0, tk.END)

cliente = socket.socket()
cliente.connect(("localhost", 5000))

ventana = tk.Tk()
chat = tk.Text(ventana)
chat.pack()
entrada = tk.Entry(ventana)
```

```
entrada.pack()
tk.Button(ventana, text="Enviar", command=enviar).pack()

threading.Thread(target=recibir, daemon=True).start()
ventana.mainloop()
```

Resumen:

- Estos ejemplos muestran cómo usar Tkinter para crear programas completos con diversas funcionalidades.
- Se aplican conceptos como eventos, canvas, temporizadores, archivos, audio, red y bases de datos.
- La práctica con estos programas refuerza la comprensión de todos los temas de la interfaz gráfica con Python.

Lección 19. Organización de Proyectos en Tkinter

19.1 Modularización del código

Modularizar significa dividir el código en archivos o funciones según su responsabilidad. Esto facilita el mantenimiento, la lectura y la reutilización. Por ejemplo, separar un archivo para la interfaz (`interfaz.py`), otro para la lógica (`logica.py`) y uno principal (`main.py`).

Ejemplo de estructura básica:

- main.py
- interfaz.py
- logica.py

Archivo: logica.py

```
def sumar(a, b):
    return a + b
```

Archivo: interfaz.py

```
import tkinter as tk

class Interfaz:
    def __init__(self, master, funcion_suma):
        self.funcion_suma = funcion_suma
        self.master = master
        self.entrada1 = tk.Entry(master)
        self.entrada2 = tk.Entry(master)
        self.boton = tk.Button(master, text="Sumar", command=self.sumar)
        self.resultado = tk.Label(master, text="")

        self.entrada1.pack()
        self.entrada2.pack()
        self.boton.pack()
        self.resultado.pack()

    def sumar(self):
        a = int(self.entrada1.get())
        b = int(self.entrada2.get())
        r = self.funcion_suma(a, b)
        self.resultado.config(text=f"Resultado: {r}")
```

Archivo: main.py

```
import tkinter as tk
from interfaz import Interfaz
from logica import sumar

ventana = tk.Tk()
app = Interfaz(ventana, sumar)
ventana.mainloop()
```

19.2 Separar lógica e interfaz

La lógica de la aplicación (cálculos, operaciones, validaciones) debe estar separada del diseño gráfico (botones, entradas, etiquetas). Esto permite probar funciones de forma independiente y facilita cambios en la interfaz sin alterar el funcionamiento interno.

Ejemplo:

Archivo: logica.py

```
def es_par(numero):
    return numero % 2 == 0
```

Archivo: interfaz.py

```
import tkinter as tk

class InterfazPar:
    def __init__(self, master, funcion_verificacion):
        self.funcion = funcion_verificacion
        self.master = master
        self.entrada = tk.Entry(master)
        self.boton = tk.Button(master, text="Verificar", command=self.verificar)
        self.resultado = tk.Label(master, text="")

        self.entrada.pack()
        self.boton.pack()
        self.resultado.pack()

    def verificar(self):
        n = int(self.entrada.get())
        if self.funcion(n):
            self.resultado.config(text="Es par")
        else:
            self.resultado.config(text="Es impar")
```

19.3 Manejo de múltiples ventanas

En proyectos más grandes, se pueden usar varias ventanas (ventanas hijas) para distintas funciones como configuración, ayuda, etc.

```
import tkinter as tk

def abrir_ventana_secundaria():
    nueva = tk.Toplevel()
    nueva.title("Ventana secundaria")
    tk.Label(nueva, text="Otra ventana").pack()

ventana = tk.Tk()
ventana.title("Principal")
tk.Button(ventana, text="Abrir otra ventana",
command=abrir_ventana_secundaria).pack()
ventana.mainloop()
```

Se recomienda encapsular cada ventana en su propia clase para facilitar la gestión.

```
class VentanaSecundaria(tk.Toplevel):
    def __init__(self, master=None):
        super().__init__(master)
        self.title("Secundaria")
        tk.Label(self, text="Hola").pack()
```

19.4 Buenas prácticas de mantenimiento

- Nombres descriptivos para variables, funciones y clases.
- Comentarios claros y docstrings para funciones.

- Uso de convenciones de estilo como PEP8.
- Manejo de errores con try-except.
- Dividir el proyecto en módulos (archivos) y paquetes (carpetas con `__init__.py`).
- Uso de archivos de configuración (`config.py`) o `.json`.
- Documentación del proyecto y del flujo de ejecución.
- Separar recursos estáticos (imágenes, archivos de datos) en carpetas.

Ejemplo de estructura de proyecto:

```
mi_app/
│
├── main.py
├── interfaz.py
├── logica.py
├── config.py
├── recursos/
│   ├── logo.png
│   └── datos.json
└── ventanas/
    ├── ventana_config.py
    └── ventana_ayuda.py
```

Resumen:

- Organizar bien el proyecto mejora su legibilidad, mantenimiento y escalabilidad.
- Separar interfaz y lógica ayuda a reutilizar y testear componentes.
- Manejar múltiples ventanas permite estructuras más complejas.
- Aplicar buenas prácticas asegura estabilidad y facilidad de actualización.

Lección 20. Personalización y Estética en Tkinter

20.1 Colores, fuentes y bordes

En Tkinter se puede personalizar el aspecto de los widgets usando opciones como `bg` (color de fondo), `fg` (color del texto), `font` (fuente), `bd` (borde), `relief` (tipo de borde) y más.

Ejemplo básico:

```
import tkinter as tk

ventana = tk.Tk()
ventana.title("Estética Básica")

etiqueta = tk.Label(ventana, text="Texto personalizado", bg="lightblue",
fg="darkblue", font=("Arial", 14, "bold"))
etiqueta.pack(padx=10, pady=10)

boton = tk.Button(ventana, text="Aceptar", bg="lightgreen", font=("Helvetica",
12), bd=4, relief="raised")
boton.pack(padx=10, pady=10)

ventana.mainloop()
```

Tipos de `relief`: flat, raised, sunken, groove, ridge.

20.2 Temas en ttk

Tkinter.ttk permite el uso de temas para cambiar automáticamente el estilo de los widgets compatibles (`ttk.Button`, `ttk.Label`, etc.). Los temas disponibles dependen del sistema operativo.

Ejemplo para ver y usar temas:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
ventana.title("Temas ttk")

estilo = ttk.Style()
temas_disponibles = estilo.theme_names()
print("Temas disponibles:", temas_disponibles)

estilo.theme_use("clam")  # Otros: 'alt', 'default', 'classic'

ttk.Label(ventana, text="Etiqueta ttk con tema").pack(pady=10)
ttk.Button(ventana, text="Botón ttk").pack(pady=10)

ventana.mainloop()
```

20.3 Aplicar estilos personalizados

Se pueden crear estilos personalizados usando `ttk.Style()` y aplicarlos a widgets `ttk`.

Ejemplo:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()
```

```
ventana.title("Estilo Personalizado")

style = ttk.Style()
style.configure("BotonGrande.TButton", font=("Verdana", 14), foreground="white",
background="#3333cc", padding=10)

ttk.Button(ventana, text="Botón Estilizado",
style="BotonGrande.TButton").pack(pady=20)

ventana.mainloop()
```

Se pueden definir diferentes estilos para distintos propósitos y aplicar temas según la plataforma o preferencia.

20.4 Aplicaciones con apariencia moderna

Para lograr una apariencia moderna se pueden aplicar las siguientes técnicas:

- Uso de `ttk` con temas modernos.
- Añadir íconos y logos en formatos `.png`.
- Bordes suaves, espaciados adecuados y paletas de colores elegantes.
- Uso de librerías como `ttkthemes` o `ttkbootstrap`.

Ejemplo usando ttkbootstrap:

```
import ttkbootstrap as tb
from ttkbootstrap.constants import *

ventana = tb.Window(themename="darkly")
ventana.title("Interfaz Moderna")

tb.Label(ventana, text="Bienvenido", font=("Arial", 16)).pack(pady=10)
tb.Entry(ventana).pack(pady=10)
tb.Button(ventana, text="Aceptar", bootstyle=SUCCESS).pack(pady=10)

ventana.mainloop()
```

Para usar ttkbootstrap se instala con `pip install ttkbootstrap`.

También se puede mejorar la estética general:

- Ajustando `padding` entre widgets.
- Centrando los contenidos con `.pack()` o `.grid()`.
- Usando imágenes y efectos visuales en `Canvas`.
- Evitando saturación de colores o estilos.

Resumen:

- Personalizar colores, fuentes y bordes mejora la experiencia visual.
- Los temas ttk permiten dar un estilo coherente a toda la interfaz.
- Se pueden crear estilos propios y reutilizarlos.
- Las interfaces modernas usan espaciado, consistencia y simplicidad visual.

Lección 21. Empaquetado y Distribución de Aplicaciones Tkinter

21.1 Empaquetar con pyinstaller o cx_Freeze

Para distribuir una aplicación de Tkinter sin requerir que el usuario tenga Python instalado, se puede usar herramientas como `pyinstaller` o `cx_Freeze` para generar ejecutables.

Usar PyInstaller:

Instalación:

```
pip install pyinstaller
```

Crear un ejecutable simple:

```
pyinstaller --onefile mi_aplicacion.py
```

Crear ejecutable con ventana oculta (sin consola):

```
pyinstaller --onefile --noconsole mi_aplicacion.py
```

Para aplicaciones con múltiples archivos (imágenes, sonidos), usar:

```
pyinstaller --onefile --add-data "imagenes/logo.png;imagenes" mi_aplicacion.py
```

En Windows, usar `;` para separar rutas; en Linux/Mac, usar `:`.

Usar cx_Freeze:

Instalación:

```
pip install cx_Freeze
```

Crear archivo `setup.py`:

```
from cx_Freeze import setup, Executable

setup(
    name="MiAppTkinter",
    version="1.0",
    description="Aplicación con Tkinter",
    executables=[Executable("mi_aplicacion.py")],
)
```

Crear ejecutable:

```
python setup.py build
```

21.2 Crear ejecutables en Windows/Linux/Mac

Windows:

- Usar PyInstaller con `--noconsole` para ocultar la consola.
- El ejecutable generado estará en la carpeta `/dist`.

Linux:

- El ejecutable generado solo funciona en sistemas similares (misma arquitectura y distribución).

- Usar AppImage o `pyinstaller` con rutas relativas.

Mac:

- Se puede usar `py2app` (alternativa a `pyinstaller` para Mac):

```
pip install py2app
python setup.py py2app
```

Para portabilidad entre sistemas operativos, se recomienda crear los ejecutables en el sistema operativo objetivo.

21.3 Incluir imágenes, iconos y recursos

Para incluir recursos (iconos, imágenes, archivos de audio, etc.):

- Crear una estructura de carpetas clara:

```
mi_proyecto/
├── recursos/
│   ├── logo.png
│   └── fondo.jpg
├── mi_aplicacion.py
```

- Usar rutas relativas dentro del código:

```
import os

ruta_base = os.path.dirname(__file__)
ruta_logo = os.path.join(ruta_base, "recursos", "logo.png")
```

- Agregar recursos al ejecutable con PyInstaller:

```
pyinstaller --onefile --add-data "recursos/logo.png;recursos" mi_aplicacion.py
```

- Para íconos en Windows, usar `.ico`:

```
pyinstaller --onefile --icon=icono.ico mi_aplicacion.py
```

21.4 Crear instaladores

Inno Setup (Windows):

- Descarga desde: https://jrsoftware.org/isinfo.php
- Crear un script `.iss` que incluya archivos del ejecutable, recursos, íconos, etc.
- Permite crear instaladores `.exe` personalizados.

NSIS (Nullsoft):

- Página oficial: https://nsis.sourceforge.io/
- También permite crear instaladores con asistentes, condiciones de instalación, y desinstaladores.

Linux:

- Empaquetar con `deb`, `rpm` o `AppImage`.
- AppImage es el más portable (funciona como un ejecutable independiente).

MacOS:

- Usar `py2app` para generar una app.
- Luego empaquetar como `.dmg` o usar herramientas de terceros como `create-dmg`.

Resumen:

- PyInstaller es la herramienta más usada para generar ejecutables multiplataforma.
- Es posible incluir recursos como imágenes e íconos fácilmente.
- La creación de instaladores mejora la presentación y facilidad de instalación para el usuario final.
- Es recomendable testear el ejecutable en el sistema de destino antes de distribuir.

Lección 22. Recursos Adicionales para Tkinter

22.1 Documentación oficial

La mejor fuente para aprender y consultar Tkinter es su documentación oficial, que incluye descripciones de widgets, métodos y ejemplos.

- Documentación oficial de Tkinter en Python:
  https://docs.python.org/3/library/tk.html
  Contiene la descripción de todos los widgets, eventos, opciones y más.
- Tkinter Wiki (no oficial pero útil):
  https://tkdocs.com/
  Tiene tutoriales modernos, comparativas y ejemplos.

22.2 Libros recomendados

Para profundizar en Tkinter y desarrollo de interfaces gráficas en Python, algunos libros destacados:

- "Python GUI Programming with Tkinter" – Alan D. Moore
  Libro completo con teoría, ejemplos prácticos y proyectos.
- "Programming Python" – Mark Lutz
  Contiene un capítulo dedicado a Tkinter y otros temas avanzados.
- "Tkinter GUI Application Development Blueprints" – Bhaskar Chaudhary
  Proyectos prácticos para aprender a crear aplicaciones completas.
- "Modern Tkinter for Busy Python Developers" – Mark Roseman
  Cubre nuevas características y mejores prácticas.

22.3 Comunidades y foros

Participar en comunidades ayuda a resolver dudas, compartir proyectos y aprender de otros.

- Stack Overflow – Etiqueta [tkinter]
  https://stackoverflow.com/questions/tagged/tkinter
  Lugar muy activo para preguntas técnicas.
- Reddit – r/learnpython y r/Python
  https://www.reddit.com/r/learnpython/
  https://www.reddit.com/r/Python/
  Sección dedicada a GUI y Tkinter.
- Foro oficial de Python
  https://discuss.python.org/c/pythontkinter/
  Comunidad oficial para discutir sobre Tkinter y herramientas relacionadas.
- Discord y Slack de Python
  En varios servidores hay canales específicos para GUI y Tkinter.

22.4 Repositorios con ejemplos

Explorar código real ayuda a entender mejores prácticas y usos avanzados.

- Repositorio oficial de ejemplos de Tkinter (en GitHub): https://github.com/python/cpython/tree/main/Demo/tkinter
- Tkinter Examples por John Shipman: https://github.com/jshipman/tkinter-examples
- Awesome Tkinter (colección de recursos y ejemplos): https://github.com/ParthJadhav/Tkinter-Tutorials
- Ejemplos con ttkbootstrap: https://github.com/israel-dryer/ttkbootstrap
- Proyectos en GitHub buscando "Tkinter GUI": https://github.com/search?q=tkinter+gui

Resumen:

- La documentación oficial es la fuente primaria para información precisa.
- Los libros ofrecen una visión profunda y ordenada para aprender.
- Las comunidades permiten compartir y resolver dudas en tiempo real.
- Los repositorios con ejemplos prácticos son vitales para entender el código real y avanzar en proyectos.

30 ejercicios básicos de Tkinter en Python con su solución y explicación

---

EJERCICIO 1
Crear una ventana básica con título y tamaño fijo.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
ventana.title("Ventana Básica")
ventana.geometry("300x200")
ventana.mainloop()
```

EXPLICACIÓN:
Se importa tkinter, se crea una ventana, se le asigna título y tamaño, y se inicia el bucle principal.

EJERCICIO 2
Agregar una etiqueta a una ventana.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
ventana.title("Etiqueta")
etiqueta = tk.Label(ventana, text="Hola Mundo")
etiqueta.pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se crea una etiqueta con texto y se empaqueta en la ventana.

EJERCICIO 3
Agregar un botón que cierra la ventana.

SOLUCIÓN:

```
import tkinter as tk

def cerrar():
    ventana.destroy()

ventana = tk.Tk()
boton = tk.Button(ventana, text="Cerrar", command=cerrar)
boton.pack()
ventana.mainloop()
```

EXPLICACIÓN:
El botón ejecuta la función `cerrar`, que destruye la ventana.

EJERCICIO 4
Mostrar un campo de entrada (Entry).

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se agrega un widget de entrada de texto.

EJERCICIO 5
Mostrar el texto ingresado al presionar un botón.

SOLUCIÓN:

```
import tkinter as tk

def mostrar():
    texto = entrada.get()
    etiqueta.config(text=texto)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
boton = tk.Button(ventana, text="Mostrar", command=mostrar)
boton.pack()
etiqueta = tk.Label(ventana)
etiqueta.pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se obtiene el texto del Entry y se muestra en una etiqueta.

EJERCICIO 6
Cambiar el color de fondo de la ventana.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
ventana.configure(bg="lightblue")
ventana.mainloop()
```

EXPLICACIÓN:
Se usa `configure` para cambiar el color de fondo.

EJERCICIO 7
Crear múltiples etiquetas con distintos textos.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
tk.Label(ventana, text="Etiqueta 1").pack()
tk.Label(ventana, text="Etiqueta 2").pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se crean varias etiquetas y se empaquetan en orden.

EJERCICIO 8
Agregar un botón que cambia el texto de una etiqueta.

SOLUCIÓN:

```
import tkinter as tk

def cambiar_texto():
    etiqueta.config(text="Texto cambiado")

ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="Texto original")
etiqueta.pack()
tk.Button(ventana, text="Cambiar", command=cambiar_texto).pack()
ventana.mainloop()
```

EXPLICACIÓN:
El botón modifica el texto de la etiqueta al ser presionado.

EJERCICIO 9
Usar el widget `Frame` para agrupar botones.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
marco = tk.Frame(ventana)
marco.pack()
tk.Button(marco, text="Botón 1").pack(side="left")
tk.Button(marco, text="Botón 2").pack(side="left")
ventana.mainloop()
```

EXPLICACIÓN:
Se usa un `Frame` para agrupar elementos visualmente.

EJERCICIO 10
Crear un cuadro de texto multilínea (Text).

SOLUCIÓN:

```
import tkinter as tk
```

```
ventana = tk.Tk()
texto = tk.Text(ventana, height=5, width=30)
texto.pack()
ventana.mainloop()
```

EXPLICACIÓN:

El widget `Text` permite entradas de varias líneas.

EJERCICIO 11

Limpiar un campo de entrada con un botón.

SOLUCIÓN:

```
import tkinter as tk

def limpiar():
    entrada.delete(0, tk.END)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Limpiar", command=limpiar).pack()
ventana.mainloop()
```

EXPLICACIÓN:

`delete` borra el contenido del Entry desde el índice 0 hasta el final.

EJERCICIO 12

Crear una lista desplegable con opciones.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
opciones = ["Rojo", "Verde", "Azul"]
variable = tk.StringVar(ventana)
variable.set(opciones[0])
menu = tk.OptionMenu(ventana, variable, *opciones)
menu.pack()
ventana.mainloop()
```

EXPLICACIÓN:

`OptionMenu` permite seleccionar una opción de un menú desplegable.

EJERCICIO 13

Mostrar la opción seleccionada de un OptionMenu.

SOLUCIÓN:

```
import tkinter as tk

def mostrar_opcion():
    etiqueta.config(text=variable.get())
```

```
ventana = tk.Tk()
variable = tk.StringVar(ventana)
variable.set("Uno")
menu = tk.OptionMenu(ventana, variable, "Uno", "Dos", "Tres")
menu.pack()
tk.Button(ventana, text="Mostrar", command=mostrar_opcion).pack()
etiqueta = tk.Label(ventana)
etiqueta.pack()
ventana.mainloop()
```

EXPLICACIÓN:

Se accede al valor del OptionMenu usando `get`.

EJERCICIO 14

Agregar botones de selección (Radiobutton).

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
opcion = tk.StringVar()

tk.Radiobutton(ventana, text="A", variable=opcion, value="A").pack()
tk.Radiobutton(ventana, text="B", variable=opcion, value="B").pack()
ventana.mainloop()
```

EXPLICACIÓN:

Los Radiobutton permiten seleccionar una sola opción.

EJERCICIO 15

Mostrar selección de Radiobutton.

SOLUCIÓN:

```
import tkinter as tk

def mostrar():
    etiqueta.config(text=opcion.get())

ventana = tk.Tk()
opcion = tk.StringVar()
tk.Radiobutton(ventana, text="Python", variable=opcion, value="Python").pack()
tk.Radiobutton(ventana, text="Java", variable=opcion, value="Java").pack()
tk.Button(ventana, text="Mostrar", command=mostrar).pack()
etiqueta = tk.Label(ventana)
etiqueta.pack()
ventana.mainloop()
```

EXPLICACIÓN:

`get` devuelve la opción seleccionada.

EJERCICIO 16

Agregar casillas de verificación (Checkbutton).

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
var1 = tk.BooleanVar()
var2 = tk.BooleanVar()

tk.Checkbutton(ventana, text="Opción A", variable=var1).pack()
tk.Checkbutton(ventana, text="Opción B", variable=var2).pack()
ventana.mainloop()
```

EXPLICACIÓN:
Las Checkbutton permiten marcar varias opciones.

EJERCICIO 17
Mostrar estado de checkbuttons.

SOLUCIÓN:

```
import tkinter as tk

def mostrar():
    resultado = ""
    if var1.get():
        resultado += "A "
    if var2.get():
        resultado += "B"
    etiqueta.config(text=resultado)

ventana = tk.Tk()
var1 = tk.BooleanVar()
var2 = tk.BooleanVar()

tk.Checkbutton(ventana, text="A", variable=var1).pack()
tk.Checkbutton(ventana, text="B", variable=var2).pack()
tk.Button(ventana, text="Mostrar", command=mostrar).pack()
etiqueta = tk.Label(ventana)
etiqueta.pack()
ventana.mainloop()
```

EXPLICACIÓN:
`get()` devuelve `True` si está marcada.

EJERCICIO 18
Cambiar el tamaño de fuente de una etiqueta.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="Texto Grande", font=("Arial", 20))
etiqueta.pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se puede definir la fuente y el tamaño con el parámetro `font`.

EJERCICIO 19
Usar `place` para posicionar widgets.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
tk.Label(ventana, text="Nombre:").place(x=10, y=20)
tk.Entry(ventana).place(x=80, y=20)
ventana.geometry("300x100")
ventana.mainloop()
```

EXPLICACIÓN:
`place` posiciona los elementos por coordenadas.

EJERCICIO 20
Contar clics en un botón.

SOLUCIÓN:

```
import tkinter as tk

contador = 0

def contar():
    global contador
    contador += 1
    etiqueta.config(text=str(contador))

ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="0")
etiqueta.pack()
tk.Button(ventana, text="Clic", command=contar).pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se usa una variable global para contar las veces que se presiona el botón.

EJERCICIO 21
Abrir una ventana secundaria al presionar un botón.

SOLUCIÓN:

```
import tkinter as tk


def abrir_ventana():

    nueva = tk.Toplevel()

    nueva.title("Secundaria")
```

```
    tk.Label(nueva, text="Ventana nueva").pack()


ventana = tk.Tk()
tk.Button(ventana, text="Abrir ventana", command=abrir_ventana).pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se crea una ventana secundaria con `Toplevel`.

EJERCICIO 22
Cerrar ventana secundaria desde sí misma.

SOLUCIÓN:

```
import tkinter as tk


def abrir():
    nueva = tk.Toplevel()
    tk.Button(nueva, text="Cerrar", command=nueva.destroy).pack()


ventana = tk.Tk()
tk.Button(ventana, text="Abrir", command=abrir).pack()
ventana.mainloop()
```

EXPLICACIÓN:
El botón dentro de la ventana secundaria llama a `destroy` sobre sí misma.

EJERCICIO 23
Crear una barra de menú básica.

SOLUCIÓN:

```
import tkinter as tk


def salir():
    ventana.quit()


ventana = tk.Tk()
menubar = tk.Menu(ventana)
archivo = tk.Menu(menubar, tearoff=0)
archivo.add_command(label="Salir", command=salir)
menubar.add_cascade(label="Archivo", menu=archivo)
ventana.config(menu=menubar)
```

```
ventana.mainloop()
```

EXPLICACIÓN:
Se usa `Menu` para crear una barra con un ítem que cierra la aplicación.

EJERCICIO 24
Agregar un mensaje emergente de información.

SOLUCIÓN:

```
import tkinter as tk
from tkinter import messagebox


def mostrar_info():
    messagebox.showinfo("Info", "Esto es un mensaje")


ventana = tk.Tk()
tk.Button(ventana, text="Mostrar mensaje", command=mostrar_info).pack()
ventana.mainloop()
```

EXPLICACIÓN:
`messagebox.showinfo` muestra un cuadro de información.

EJERCICIO 25
Cambiar el color de una etiqueta al presionar un botón.

SOLUCIÓN:

```
import tkinter as tk


def cambiar_color():
    etiqueta.config(fg="blue")


ventana = tk.Tk()
etiqueta = tk.Label(ventana, text="Texto colorido")
etiqueta.pack()
tk.Button(ventana, text="Azul", command=cambiar_color).pack()
ventana.mainloop()
```

EXPLICACIÓN:
Se usa `config(fg=...)` para cambiar el color del texto.

EJERCICIO 26
Insertar texto en un Text al pulsar un botón.

SOLUCIÓN:

```
import tkinter as tk


def insertar():
    texto.insert(tk.END, "Línea añadida\n")


ventana = tk.Tk()
texto = tk.Text(ventana)
texto.pack()
tk.Button(ventana, text="Añadir", command=insertar).pack()
ventana.mainloop()
```

EXPLICACIÓN:
`insert` coloca texto en el widget `Text`.

EJERCICIO 27
Mostrar el contenido del Text en una etiqueta.

SOLUCIÓN:

```
import tkinter as tk


def mostrar():
    contenido = texto.get("1.0", tk.END)
    etiqueta.config(text=contenido)


ventana = tk.Tk()
texto = tk.Text(ventana, height=4, width=30)
texto.pack()
tk.Button(ventana, text="Mostrar", command=mostrar).pack()
etiqueta = tk.Label(ventana)
etiqueta.pack()
ventana.mainloop()
```

EXPLICACIÓN:
`get("1.0", END)` obtiene todo el contenido del widget `Text`.

EJERCICIO 28
Usar Scrollbar con Text.

SOLUCIÓN:

```
import tkinter as tk
```

```
ventana = tk.Tk()
scroll = tk.Scrollbar(ventana)
scroll.pack(side=tk.RIGHT, fill=tk.Y)

texto = tk.Text(ventana, yscrollcommand=scroll.set)
texto.pack(side=tk.LEFT)
scroll.config(command=texto.yview)
ventana.mainloop()
```

EXPLICACIÓN:
Se vincula un `Scrollbar` con un `Text` para desplazar el contenido.

EJERCICIO 29
Crear botones con imágenes.

SOLUCIÓN:

```
import tkinter as tk

ventana = tk.Tk()
imagen = tk.PhotoImage(width=50, height=50)
boton = tk.Button(ventana, image=imagen)
boton.pack()
ventana.mainloop()
```

EXPLICACIÓN:
`PhotoImage` permite cargar imágenes, incluso vacías, en botones.

EJERCICIO 30
Hacer que un botón desactive otro.

SOLUCIÓN:

```
import tkinter as tk

def desactivar():
    boton2.config(state="disabled")

ventana = tk.Tk()
boton1 = tk.Button(ventana, text="Desactivar B2", command=desactivar)
boton1.pack()
boton2 = tk.Button(ventana, text="Botón 2")
```

```
boton2.pack()

ventana.mainloop()
```

EXPLICACIÓN:
Usando `config(state="disabled")` se desactiva un botón.

¿Quieres que prepare más ejercicios de nivel intermedio o deseas estos 30 en un archivo TXT o PDF?

Agenda completa json en tkinter

Programa completo en **Tkinter** que permite:

- **Agregar** fichas con nombre, dirección y teléfono.
- **Consultar** una ficha por nombre.
- **Listar todas** las fichas.
- **Borrar** una ficha por nombre.
- Los datos se guardan en un archivo `fichas.json`.

```
import tkinter as tk
from tkinter import messagebox
import json
import os

ARCHIVO = "fichas.json"

def cargar_fichas():
    if os.path.exists(ARCHIVO):
        with open(ARCHIVO, "r") as f:
            return json.load(f)
    return {}

def guardar_fichas(fichas):
    with open(ARCHIVO, "w") as f:
        json.dump(fichas, f, indent=4)

def agregar_ficha():
    nombre = entrada_nombre.get()
    direccion = entrada_direccion.get()
    telefono = entrada_telefono.get()
    if nombre:
        fichas[nombre] = {"direccion": direccion, "telefono": telefono}
        guardar_fichas(fichas)
        messagebox.showinfo("Éxito", "Ficha agregada correctamente.")
        entrada_nombre.delete(0, tk.END)
        entrada_direccion.delete(0, tk.END)
        entrada_telefono.delete(0, tk.END)
    else:
        messagebox.showwarning("Error", "El campo 'nombre' es obligatorio.")

def consultar_ficha():
    nombre = entrada_nombre.get()
    if nombre in fichas:
        ficha = fichas[nombre]
        mensaje = f"Nombre: {nombre}\nDirección: {ficha['direccion']}\nTeléfono:
{ficha['telefono']}"
        messagebox.showinfo("Ficha encontrada", mensaje)
    else:
        messagebox.showwarning("No encontrada", "No existe una ficha con ese
nombre.")

def listar_fichas():
    if fichas:
        mensaje = ""
        for nombre, datos in fichas.items():
            mensaje += f"Nombre: {nombre}\nDirección: {datos['direccion']}
\nTeléfono: {datos['telefono']}\n\n"
        messagebox.showinfo("Todas las fichas", mensaje)
```

```
    else:
        messagebox.showinfo("Sin datos", "No hay fichas guardadas.")

def borrar_ficha():
    nombre = entrada_nombre.get()
    if nombre in fichas:
        del fichas[nombre]
        guardar_fichas(fichas)
        messagebox.showinfo("Eliminada", "Ficha borrada correctamente.")
    else:
        messagebox.showwarning("No encontrada", "No existe una ficha con ese
nombre.")

# Cargar fichas al iniciar
fichas = cargar_fichas()

# Crear ventana principal
ventana = tk.Tk()
ventana.title("Gestor de Fichas JSON")

# Etiquetas y entradas
tk.Label(ventana, text="Nombre:").grid(row=0, column=0)
entrada_nombre = tk.Entry(ventana)
entrada_nombre.grid(row=0, column=1)

tk.Label(ventana, text="Dirección:").grid(row=1, column=0)
entrada_direccion = tk.Entry(ventana)
entrada_direccion.grid(row=1, column=1)

tk.Label(ventana, text="Teléfono:").grid(row=2, column=0)
entrada_telefono = tk.Entry(ventana)
entrada_telefono.grid(row=2, column=1)

# Menú
menu = tk.Menu(ventana)
ventana.config(menu=menu)

menu_fichas = tk.Menu(menu, tearoff=0)
menu.add_cascade(label="Opciones", menu=menu_fichas)
menu_fichas.add_command(label="Agregar ficha", command=agregar_ficha)
menu_fichas.add_command(label="Consultar ficha por nombre",
command=consultar_ficha)
menu_fichas.add_command(label="Listar todas las fichas", command=listar_fichas)
menu_fichas.add_command(label="Borrar ficha por nombre", command=borrar_ficha)
menu_fichas.add_separator()
menu_fichas.add_command(label="Salir", command=ventana.quit)

ventana.mainloop()
```

Epílogo

Al llegar al final de este libro, solo puedo decir que este viaje con Tkinter ha sido mucho más que una inmersión en un módulo de programación. Ha sido una experiencia de descubrimiento, de paciencia, de errores y aciertos, y sobre todo, de compromiso con una idea: que el conocimiento debe estar al alcance de todos.

Tkinter, a pesar de su apariencia sencilla, encierra un potencial inmenso. A través de sus ventanas, botones, eventos y diseños, hemos visto cómo una interfaz gráfica no solo comunica con el usuario, sino que también expresa la lógica y la creatividad de quien la construye. Espero que este libro haya cumplido con su propósito de ayudarte a entenderlo, dominarlo y, por qué no, disfrutarlo.

Pero más allá del código y los widgets, quiero recordarte algo esencial: programar no es solo escribir instrucciones para una máquina. Es también un acto de creación, de pensamiento crítico, de libertad. Y si algo deseo dejar como última idea en estas páginas es que nunca dejes de aprender, nunca dejes de compartir lo aprendido y, sobre todo, nunca dejes que nadie te diga que el conocimiento no es para ti.

Este libro es solo un punto de partida. Desde aquí, el camino continúa. Con nuevas ideas, nuevos proyectos y nuevas preguntas que te impulsarán a seguir creciendo.

Gracias por haber llegado hasta aquí. Si estas páginas han encendido en ti una chispa de curiosidad o una voluntad de crear, entonces todo este esfuerzo ha valido la pena.

Nos vemos en el próximo proyecto.
